Università degli Studi di Pisa
Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica

Analisi e valutazione di algoritmi distribuiti per la costruzione della Triangolazione di Delaunay

Candidato:
Giovanni Catania

Relatori:
Prof. Laura Ricci
Dott. Barbara Guidi

Controrelatore:
Prof. Stefano Chessa

Anno Accademico 2014/2015

Abstract
Le triangolazioni di Delaunay sono molto utili grazie alle loro proprietà matematiche, che vengono sfruttate per definire overlay utilizzati in molte applicazioni distribuite, da reti peer-to-peer a reti di sensori e geografiche. Per queste ragioni, recentemente sono stati proposti molti algoritmi distribuiti per la costruzione di overlay basati sulla triangolazione di Delaunay. La tesi presenta una rassegna dei principali algoritmi distribuiti persentati negli ultimi anni, e dei diversi contesti in cui tali algoritmi sono utilizzati, con particolare attenzione alle tecniche innovative. L'analisi ha portato alla definizione di New ACE, un nuovo algoritmo distribuito, che è stato confrontato con due diverse tecniche di stato dell'arte. La tesi presenta un insieme di risultati sperimentali che mostrano i pro e i contro di questi algoritmi.

Indice

1 Introduzione **2**

1.1 Struttura della tesi 5

2 La Triangolazione di Delaunay **7**

2.1 La Triangolazione di Delaunay 8

2.2 Algoritmi per la costruzione della Triangolazione di Delaunay 14

2.2.1 Il Flip Algorithm 14

2.2.2 Algoritmo di Bowyer-Watson 21

2.2.3 Algoritmi Divide et Impera 25

3 Algoritmi distribuiti per Delaunay overlay **30**

3.1 Algoritmi orientati al routing geografico 32

3.1.1 GeoPeer 33

3.1.2 Skip Delaunay Network 38

3.2 Algoritmi localizzati per la costruzione della triangolazione di Delaunay . . . 40

3.2.1 Planar Localized Delaunay Triangulation di Li et al. 41

3.2.2 Fast Localized Delaunay Triangulation 44

3.2.3 Restricted Delaunay Graph 47

3.2.4 Conclusioni 50

3.3 Algoritmi di supporto agli Ambienti Virtuali Distribuiti 51

3.3.1 L'algoritmo di Buyukkaya et al. 51

3.3.2 L'algoritmo di Ghaffari et al. 53

3.3.3 Conclusioni 56

3.4 Algoritmi orientati a reti distribuite e P2P 57

3.4.1 L'algoritmo distribuito di J. Liebeherr et al. 57

3.4.2 L'algoritmo dinamico distribuito in d dimensioni di M. Steiner et al. . 62

3.4.3 L'algoritmo incrementale distribuito di M. Ohnishi et al. per Virtual Collaborative Spaces 66

3.4.4 Conclusioni 69

3.5 Algoritmi distribuiti di supporto a problemi di Copertura di Aree 70
3.5.1 Delaunay Triangulation Score 70
3.5.2 Improved Delaunay Triangulation 71

4 New ACE: un nuovo protocollo per la Delaunay distribuita 75
4.1 Il protocollo GoDel 77
4.1.1 Il livello GoDel 78
4.1.2 L'algoritmo 80
4.2 Il protocollo ACE 81
4.2.1 Il protocollo ACE join 87
4.2.2 Correttezza della procedura di Join sequenziale 89
4.2.3 Il protocollo ACE leave 91
4.2.4 Il protocollo ACE failure 92
4.2.5 Il protocollo di Manutenzione ACE 93
4.3 Osservazioni sulla correttezza di ACE 95
4.4 La suite di protocolli New ACE 100

5 New ACE: Implementazione 107
5.1 PeerSim 107
5.2 Calcolo della triangolazione di Delaunay 109
5.3 Classi utilizzate 109
5.3.1 DTTransport e DDTNode 112
5.3.2 Messaggi 112
5.3.3 GreedyNearestNeighbor 114
5.3.4 ACELinkable 115
5.3.5 ACEProtocol 116
5.3.6 InetCoordinates 117
5.4 Implementazione del protocollo di manutenzione 117

6 Risultati sperimentali 119
6.1 Convergenza di ACE e New ACE per Join Sequenziali 120
6.2 Numero di messaggi per Join Sequenziali 122
6.3 Leave sequenziali 124
6.4 Test su Dataset Reale con churn 125
6.5 Test sul tempo di convergenza 129
6.6 Considerazioni finali 132

7 Conclusioni 133

Bibliografia **135**

Capitolo 1

Introduzione

Negli ultimi anni l'utilizzo di servizi *location-aware*[10] è aumentato notevolmente nel settore informatico. Si definiscono servizi *location-aware* i servizi dai quali è possibile estrarre informazioni relative alla posizione geografica per ricavarne un'utilità. Esempi di servizi location-aware sono, ad esempio, quelli che permettono di recuperare informazioni relative al monitoraggio geografico di un ambiente, oppure servizi che permettono l'invio di notifiche ad alcuni nodi in una determinata posizione, oppure di ricercare un'informazione relativa ad una determinata zona geografica (per esempio, ristoranti nelle vicinanze).

Focalizzandoci sul monitoraggio ambientale, tale servizio è in genere realizzato tramite reti di sensori wireless, le quali devono essere in grado di supportare algoritmi di routing efficienti per l'invio delle informazioni relative al monitoraggio all'unità incaricata di raccoglierle. D'altro canto, queste applicazioni devono fare i conti con dei problemi reali, quali la limitazione data dall'energia residua dei sensori e dal raggio di trasmissione limitato di questi.

Nasce quindi l'esigenza di una struttura che supporti il *routing geografico*, cioè un routing che si basi sulle posizioni geografiche dei nodi nella rete, che sia di facile costruzione in maniera distribuita con un numero limitato di messaggi, per limitare l'utilizzo di eccessiva energia residua nei sensori, e che possa essere costruito tramite *algoritmi localizzati*, cioè tramite algoritmi distribuiti che basino le loro scelte solamente sulla conoscenza locale del nodo che li esegue.

Molte overlay geografiche basano la propria struttura sulla *Triangolazione di Delaunay* [6]. La triangolazione di Delaunay è una struttura matematica con numerose proprietà che rendono l'overlay costruita su questa adatta al *routing geografico*. Inoltre, l'overlay di Delaunay supporta algoritmi di routing semplici ed efficaci con un numero limitato di messaggi, quali il *compass routing* [20], che sceglie il nodo a cui inviare il messaggio sulla base dell'angolo, e il *greedy routing*, che sceglie il nodo a cui inviare il messaggio sulla base della distanza euclidea.

Solitamente la costruzione e il mantenimento di un overlay distribuito basato sulla triangolazione di Delaunay risulta costosa in termini del numero di messaggi. Quindi, sono state proposte numerose soluzioni in letteratura che, in un qualche modo, limitassero il numero di messaggi necessari per tale costruzione. Alcune di queste soluzioni propongono la costruzione della triangolazione di Delaunay completa sfruttando le proprietà matematiche di questa per limitare il numero di messaggi, altre propongono la costruzione di una triangolazione di Delaunay approssimata o di strutture che garantiscono lo stesso supporto al routing geografico della triangolazione di Delaunay originale.

Il vantaggio di definire un algoritmo distribuito che costruisca una triangolazione di Delaunay completa consiste nel fatto che questo può essere utilizzato in molti campi applicativi, tra cui le reti P2P, gli Ambienti Virtuali Distribuiti (DVE) ed i *Virtual Collaborative Spaces*[28]. D'altro canto, il vantaggio delle soluzioni che propongono la costruzione di una triangolazione di Delaunay approssimata è che queste possono essere utilizzate efficientemente in applicazioni alle reti di sensori.

Algoritmi per la costruzione di una triangolazione di Delaunay completa sono stati proposti in diversi lavori. Lee et al. [21] presentano ACE (Accuracy Correctness Efficiency), il quale calcola la triangolazione di Delaunay utilizzando l'approccio con Candidate Set, che prevede lo scambio di una serie di messaggi tra i nodi geograficamente vicini al fine di riempire la vista locale dei nodi, limitando il numero di messaggi utilizzati restringendo la richiesta di nodi da inserire nella vista locale solamente a un nodo per ogni nuovo triangolo nella triangolazione. Liebeherr et al. [24] propongono un algoritmo distribuito per la costruzione della triangolazione di Delaunay basato sul Flip Algorithm centralizzato[9]. Ohnishi et al.[27] descrivono un algoritmo orientato ai *Virtual Collaborative Space*. Baraglia et al. propongono GoDel [5], un protocollo distribuito per la costruzione della triangolazione di Delaunay completa che utilizza protocolli di Gossip per la diffusione della conoscenza nella rete, che converge in un numero limitato e contenuto di iterazioni.

Alcuni algoritmi proposti per le reti di sensori wireless considerano la possibilità di inviare messaggi entro un certo raggio poiché, non essendo collegati tra loro tramite connessioni cablate, il loro raggio di trasmissione è limitato dalla potenza del segnale wireless. Per questo motivo, è possibile che la distanza di un sensore da un suo vicino nella triangolazione di Delaunay sia tale da impedire il contatto con tale sensore. Questa problematica è di grande rilevanza perché pone il problema di considerare alcuni casi in cui il routing geografico su una rete di sensori non può funzionare su una struttura come la triangolazione di Delaunay perché, sebbene due sensori siano vicini di Delaunay tra loro, la loro distanza fisica sia tale che non possano comunicare. Per questo motivo, sono stati proposti degli *algoritmi localizzati* per il calcolo di strutture basate su triangolazione di Delaunay con supporto al routing geografico. Gli algoritmi localizzati sono la classe di algoritmi che risolvono il problema dato utilizzando solamente l'informazione locale dei nodi, dove per informazione locale si intende

l'informazione direttamente accessibile dal nodo che lo esegue. Strutture costruibili tramite algoritmi localizzati sono state proposte in letteratura, come il Grafo di Gabriel (GG)[7] o il Relative Neighborhood Graph[36], i quali però non hanno una limitazione allo spanning ratio[1]. Con lo scopo di utilizzare nuove strutture che limitino tale rapporto, sono state proposte le strutture *Planar Localized Delaunay Triangulation*[23] e *Restricted Delaunay Graph*[12].

Le strutture approssimate basate su triangolazione di Delaunay non sono state proposte, però, solamente in applicazioni alle reti di sensori, ma anche semplicemente per porre una limitazione al numero di messaggi inviati in applicazioni richiedenti una latenza bassa di comunicazione nella rete. Un esempio di queste sono le applicazioni ad Ambienti Virtuali distribuiti (DVE).

Gli Ambienti Virtuali distribuiti sono dei veri e propri mondi virtuali che ogni utente può esplorare autonomamente e gestiti in maniera distribuita. La principale problematica da tener conto in applicazioni di questo genere è che la relativa rete è molto dinamica, poiché gli utenti si connettono, si disconnettono e cambiano la loro posizione nell'ambiente molto frequentemente.

Buyukkaya et al. [8] propongono una costruzione di un'approssimazione della triangolazione di Delaunay rilassando i suoi vincoli considerando, per ogni nodo, solo i vicini all'interno di un area locale al nodo stesso. Buyukkaya et al. forniscono inoltre un protocollo per la gestione dei movimenti dei nodi e dei cambiamenti di posizione, aggiornando la triangolazione soltanto dopo un certo spostamento dell'avatar, in modo da minimizzare i messaggi usati.

Ghaffari et al. [14] descrivono una soluzione in grado di gestire efficientemente l'aggiornamento di posizione dei nodi sfruttando algoritmi di routing greedy o compass. La triangolazione costruita dall'algoritmo presentato da Ghaffari et al., a differenza di quella di Buyukkaya et al., è completa.

Questa tesi presenta inizialmente una rassegna dei principali algoritmi distribuiti per la costruzione della triangolazione di Delaunay focalizzata sui diversi campi applicativi in cui questa è utilizzata. Successivamente, l'analisi si focalizza su due particolari protocolli che utilizzano tecniche diverse per l'acquisizione della conoscenza locale necessaria alla costruzione della triangolazione distribuita: GoDel, il quale è di grande interesse perché si distingue dagli altri algoritmi per l'utilizzo di Cyclon [39], un protocollo di Gossip che ad ogni iterazione permette lo scambio delle viste di nodi selezionati in modo casuale, e permette la diffusione della conoscenza dei nodi nella rete in poche iterazioni; ACE, il quale utilizza uno scambio di messaggi tra i nodi geograficamente vicini per il riempimento delle viste locali dei nodi, utilizzate per costruire le triangolazioni locali di questi e per il calcolo dei loro vicini di Delaunay.

[1]Si definisce *spanning ratio* di un grafo $G(N, A)$ su un insieme di punti in un piano euclideo il massimo rapporto per tutte le coppie di due dati nodi $u, v \in N$ della distanza del cammino minimo nel grafo tra u e v e la distanza euclidea tra i nodi

Il protocollo ACE prevede 4 procedure: una procedura per la fase di Join, una per la fase di Leave, una per la gestione dei Fallimenti e un protocollo di manutenzione per la correzione delle inconsistenze della rete dovute a operazioni concorrenti. In dettaglio, ogni nodo in ACE utilizza i messaggi scambiati con i nodi nelle vicinanze per riempire il proprio insieme *Candidate Set*, il quale contiene un insieme di nodi candidati a potenziali vicini. L'insieme Candidate Set viene poi utilizzato dal nodo per costruire la triangolazione di Delaunay locale. La triangolazione totale costruita in distribuito, infine, corrisponde all'unione delle triangolazioni di Delaunay locali costruite dai nodi della rete. L'analisi dell'algoritmo ACE ha portato all'individuazione di un errore nella dimostrazione della sua correttezza. Il problema rilevato consiste nel fatto che i nodi in fase di join non possono inserire nel proprio insieme Candidate Set tutti i propri vicini di Delaunay, e questo non consente la convergenza del protocollo ACE. Individuata la causa della non convergenza del protocollo, è stato definito un insieme di nuovi messaggi che includesse informazione aggiuntiva relativa ai nodi nelle vicinanze di un nodo, ed è stato utilizzato per la realizzazione di un nuovo algoritmo, chiamato *New ACE*, il quale rappresenta una soluzione nuova rispetto ad ACE perché prevede l'utilizzo di procedure diverse e nuovi messaggi.

Sono stati infine effettuati degli esperimenti su ACE, New ACE e GoDel al fine di valutare il numero di messaggi inviati, la convergenza a una soluzione corretta e il tempo di esecuzione affinché la convergenza sia raggiunta. Gli esperimenti osservati hanno messo in risalto che ACE, senza l'utilizzo di un protocollo di manutenzione, non risulta corretto, nemmeno per operazioni di join sequenziali. I test eseguiti hanno inoltre rivelato che il numero di cicli di simulazione necessari alla convergenza di ACE e New ACE dipende dalla frequenza di esecuzione del protocollo di manutenzione. Gli esperimenti sul numero di messaggi hanno rivelato che tale numero è inferiore in New ACE rispetto a GoDel, a causa dell'utilizzo di protocolli di Gossip di quest'ultimo. D'altro canto, GoDel si auto-stabilizza grazie ai protocolli di Gossip, quindi non ha necessità di un protocollo temporizzato per la riparazione della rete. Questo significa che GoDel, a differenza di ACE e New ACE, non ha bisogno di ulteriori meccanismi per la determinazione della frequenza di esecuzione di un protocollo temporizzato.

1.1 Struttura della tesi

Nel secondo capitolo viene presentato la triangolazione di Delaunay dal punto di vista geometrico e le sue principali proprietà matematiche. Viene inoltre presentata, in questo capitolo, una descrizione dei principali algoritmi per la costruzione della triangolazione di Delaunay in ambiente centralizzato. Lo stato dell'arte comprende l'analisi delle principali proprietà geometriche della triangolazione di Delaunay.

Il capitolo 3 presenta una rassegna dei principali algoritmi proposti in ambiente distribuito, divisi a seconda della loro applicazione nei campi reali. Tra questi, sono presentati algoritmi localizzati per la costruzione della triangolazione di Delaunay, con numerose applicazioni alle reti di sensori wireless. Sono presentati, inoltre, alcuni algoritmi con applicazione agli ambienti virtuali distribuiti e alle reti distribuite P2P.

Il capitolo 4 presenta l'analisi in dettaglio di due protocolli recenti per la costruzione della triangolazione di Delaunay che utilizzano tecniche diverse per il riempimento della vista locale dei nodi: il primo protocollo è GoDel (Gossip Delaunay)[5], il quale utilizza protocolli di gossip per il riempimento delle viste dei nodi, e che consente la convergenza a una soluzione corretta con un piccolo numero di iterazioni. Il secondo protocollo analizzato è ACE (Accuracy Correctness Efficiency), presentato da Lee et al. nel 2008 ([21]), il quale utilizza uno scambio di messaggi con i nodi nelle vicinanze per il riempimento delle viste locali dei nodi. La particolarità di ACE è che adotta delle metodologie interessanti per la riduzione del numero di messaggi necessari alla convergenza. Il capitolo 3 presenta successivamente un'analisi dettagliata dei Lemmi e i Teoremi di correttezza di ACE, la quale ha portato alla dimostrazione della non completa correttezza dello stesso. Quindi, da tale analisi è nata la realizzazione di New ACE, un protocollo originale basato su ACE, descritto sempre nel capitolo 4. Il capitolo 5 presenta l'implementazione di ACE e New ACE, con attenzione alle classi e alle tecniche utilizzate. Il capitolo 6 presenta un confronto delle performance di ACE, New ACE e GoDel. I test effettuati per valutare le performance mirano a valutare principalmente il numero di messaggi necessario agli algoritmi per la convergenza a una soluzione corretta, il numero di iterazioni necessarie e il comportamento dei protocolli in presenza di una rete dinamica, in cui i nodi si disconnettono in maniera concorrente.

Infine, nel capitolo 7 sono riportate le conclusioni sul lavoro di tesi svolto e i possibili sviluppi futuri.

Capitolo 2

La Triangolazione di Delaunay

La *Triangolazione di Delaunay* è una struttura geometrica molto utilizzata in ambienti distribuiti in quanto gode di particolari proprietà matematiche che ne garantiscono algoritmi semplici per problemi di routing geografico localizzato. Formalmente, dato uno spazio a due dimesioni G e un insieme di punti V in esso, una triangolazione di Delaunay consiste nella suddivisione di G in triangoli tale che i vertici di questi triangoli siano i punti in V e tale che l'angolo minimo di tali triangoli sia massimizzato.

La triangolazioni di Delaunay sono molto utilizzate in svariati ambiti, dalla geometria computazionale alle reti di sensori, alle reti Peer to Peer, dove, grazie alle loro proprietà è possibile definire algoritmi di routing che sfruttano le proprietà delle triangolazioni stesse. Proprietà matematiche delle triangolazioni di Delaunay che possono essere sfruttate per la definizione di algoritmi distribuiti:

1. per ogni triangolo T della triangolazione di Delaunay su V, $DT(V)$, il cerchio passante per i vertici di T non contiene alcun altro punto della rete. Questa proprietà viene sfruttata per risolvere problemi di *area coverage*.

2. è scalabile: ogni punto tiene in considerazione solamente i cambiamenti nelle vicinanze della sua posizione nello spazio. Inoltre, le modifiche alla struttura geometrica hanno effetti *locali* sulla triangolazione.

3. è il duale della *tassellatura di Voronoi*, molto utilizzata in ambienti virtuali distribuiti, in particolare per il calcolo delle *Aree di Interesse* dei nodi della rete.

L'utilizzo di una triangolazione di Delaunay comporta alcuni problematiche, quali:

1. un grafo costruito tramite una triangolazione di Delaunay non è uno *small world* [42], ovvero il diametro della rete non è logaritmico rispetto al numero dei nodi. In particolare, nonostante l'utilizzo di algoritmi greedy, potrebbe essere necessario un grande numero di hop per effettuare la consegna di un pacchetto da un nodo u a un nodo v posti a estremi diversi della rete.

2. la costruzione in maniera distribuita di una triangolazione di Delaunay richiede lo scambio di un numero elevato di messaggio. Questo rende difficile l'applicazione delle triangolazioni di Delaunay a problemi reali a causa di possibili limitazioni nei collegamenti o nell'energia dei sensori, oppure semplicemente perché si vogliono evitare sovraccarichi nella rete in problemi in cui si richiede una bassa latenza nella consegna dei messaggi.

Questo capitolo è strutturato come segue: la sezione 2.1 introduce i principali concetti matematici alla base della triangolazione di Delaunay, nonché i concetti matematici utilizzati negli algoritmi per la costruzione di questa; la sezione 2.2 descrive alcuni algoritmi presenti in letteratura, in particolare il *Flip Algorithm*, l'*algoritmo di Bowyer-Watson* [31] e, infine, l'*algoritmo Divide et Impera*.

2.1 La Triangolazione di Delaunay

Prima di dare la definizione di Triangolazione di Delaunay, è necessario dare una definizione formale di *Triangolazione*.

Definizione 2.1 (Massima suddivisione planare)**.** Sia $V = \{v_1, v_2, \dots, v_n\}$ un insieme di punti in un piano euclideo P_E. Si dice *Massima suddivisione planare* una suddivisione $S = \{N, E\}$ del piano in cui nessun arco può essere aggiunto senza distruggere la sua planarità.

Definizione 2.2 (Triangolazione)**.** Sia $V = \{v_1, v_2, \dots, v_n\}$ un insieme di punti in un piano euclideo P_E. Si dice *Triangolazione* la massima suddivisione planare $S = \{V, E\}$ in cui l'insieme dei vertici coincide con l'insieme dei punti del piano V.

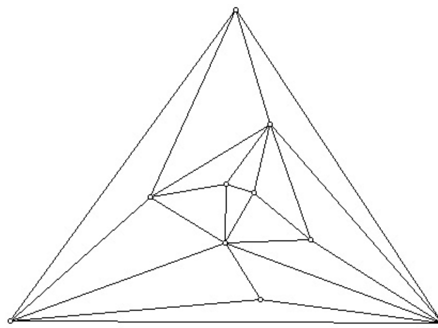

Figura 2.1: Una triangolazione di un insieme di punti in un piano bidimensionale

Poiché ogni poligono è triangolarizzabile, le triangolazioni di un insieme di punti esistono sempre. Inoltre, la triangolazione di un poligono è sempre formata da triangoli, cosa non vera se si considerano particolari configurazioni di punti, i quali vanno considerati nelle seguenti definizioni.

Definizione 2.3 (Collinearità)**.** Un insieme di punti V si dice *Collineare* se ogni punto in V risiede nella stessa retta. In questo caso, si dice che l'insieme V gode della proprietà di *Collinearità*.

Definizione 2.4 (Convex Hull)**.** Si dice *Convex Hull* di un insieme di punti V in un piano n-dimensionale l'intersezione di tutti gli insiemi convessi contenuti da V. Per un insieme di punti $v_1, v_2, \ldots, v_n$, il Convex Hull C è dato da

$$C = \{\sum_{j=1}^{n} \lambda_j v_j : \lambda_j \geq 0 \textit{ per ogni } j \textit{ e } \sum_{j=1}^{n} \lambda_j = 1\}$$

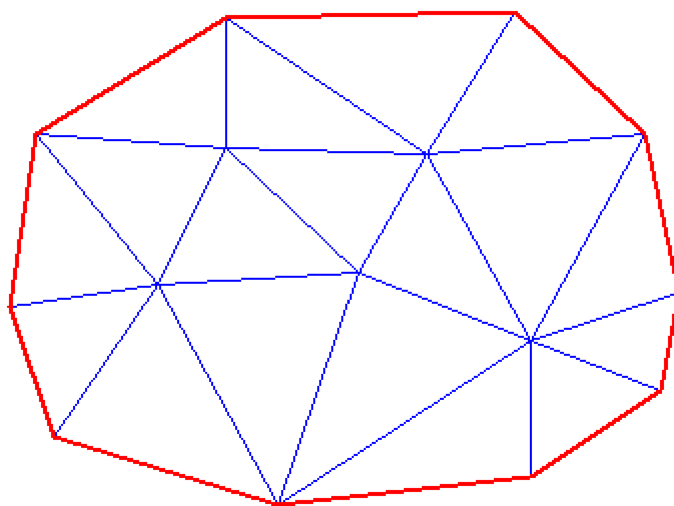

Figura 2.2: In rosso: confine del Convex hull di un insieme di punti

Il Teorema 2.1.1 ci fornisce un valore esatto del numero di triangoli e di archi un una qualunque triangolazione di un insieme di punti V, che dipende dal numero di punti del piano e dal numero di punti nel confine del Convex hull di V.

Teorema 2.1.1. *Sia V un insieme di n punti in un piano, non tutti collineari, e sia k il numero di nodi in V che giacenti sul confine del Convex hull di V. Allora, il numero di triangoli di una qualunque triangolazione $T(V)$ è $2n - k - 2$, mentre il numero di archi di $T(V)$ è $3n - k - 3$.*

È necessario specificare che non tutte le triangolazioni sono uguali: alcune di queste godono di particolari proprietà matematiche che ne permettono l'utilizzo in varie applicazioni: dalla geometria computazionale alla modellazione di terreni, dalle mesh grafiche a altro ancora.
Data una triangolazione T su un insieme di punti V, si supponga che questa sia composta da m triangoli, e si considerino i $3m$ angoli relativi agli m triangoli della triangolazione

$\alpha_{1_1}, \alpha_{1_2}, \ldots, \alpha_{m_2}, \alpha_{m_3}$. Si dispongano tali angoli all'interno di un vettore $A(T) = \{\alpha_i\}$ in ordine crescente. Utilizzando tali vettori di angoli, è possibile definire un ordinamento tra le triangolazioni T_i costruibili nell'insieme di punti V: in dettaglio, date due triangolazioni $T_1(V)$ e $T_2(V)$, diciamo che $T_1 \leq T_2$ se $A(T_1) \leq A(T_2)$, dove l'ordinamento tra vettori è definito come l'ordinamento lessicografico.
Essendo l'insieme dei punti V nel piano che stiamo considerando finito, allora anche le triangolazioni possibili in V sono finite: quindi, esisterà una triangolazione $T_o(V)$ tale che $\forall T(V) \in \{T_i(V)\}$ $A(T_o(V)) \geq A(T(V))$, dove $\{T_i(V)\}$ rappresenta l'insieme delle possibili triangolazioni nell'insieme di punti V. Tale triangolazione prende il nome di *triangolazione angolo-ottimale*.

Il seguente teorema esprime la relazione tra triangolazione angolo-ottimale e la triangolazione di Delaunay.

Teorema 2.1.2. *Sia V un insieme di punti in un piano euclideo. Allora una triangolazione $T(V)$ di V è angolo-ottimale se e soltanto se $T(V)$ è una triangolazione di Delaunay.*

Il Teorema 2.1.2 definisce la triangolazione di Delaunay $DT(V)$ di un insieme di punti V come la triangolazione che massimizza l'angolo minimo.

Nella sezione introduttiva, questa è stata descritta come il duale di un'altra suddivisione planare nota come *tassellatura di Voronoi*[4].

Definizione 2.5 (tassellatura di Voronoi)**.** Si consideri un insieme $V = \{v_1, ..., v_n\}$ di n punti in un piano euclideo P_E. Sia $d(v_i, v_j)$ la distanza euclidea tra i nodi v_i e v_j. Si dice *cella di Voronoi* $VC(v_i)$ di v_i l'insieme dei punti del piano più vicini a v_i rispetto che a tutti gli altri punti dell'insieme V, denotata con $VC(v_i) = \{x \in P_E | d(x, v_i) \leq d(x, v_j), j = 1, ..., n, j \neq i\}$.
L'insieme :

$$S = \{V(v_i) | v_i \in V\}$$

è detto *Tassellatura di Voronoi.* Il punto v_i è detto *nucleo* della cella di Voronoi $VC(v_i)$.

Un lato di una cella di Voronoi è condiviso tra 2 celle di Voronoi, ed è l'insieme dei punti equidistanti dai nuclei di tali celle. Si considerino tali lati: se si traccia la perpendicolare su questi e si collegano i nuclei v_i delle celle $VC(v_i)$, si ottiene una nuova suddivisione dello spazio: una triangolazione angolo-ottimale, e quindi una triangolazione di Delaunay sull'insieme V.

Una definizione più usata di triangolazione di Delaunay è la seguente:

Definizione 2.6 (Triangolazione di Delaunay)**.** Una *Triangolazione di Delaunay* di un insieme di punti V è una triangolazione con la proprietà che nessun punto dell'insieme V risiede all'interno della circonferenza che circoscrive un qualsiasi triangolo della triangolazione.

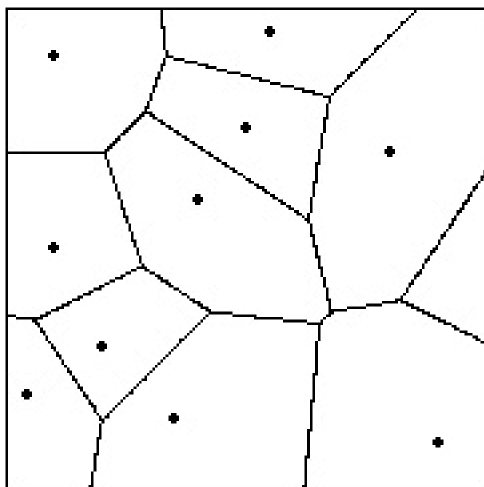

Figura 2.3: Tassellatura di Voronoi

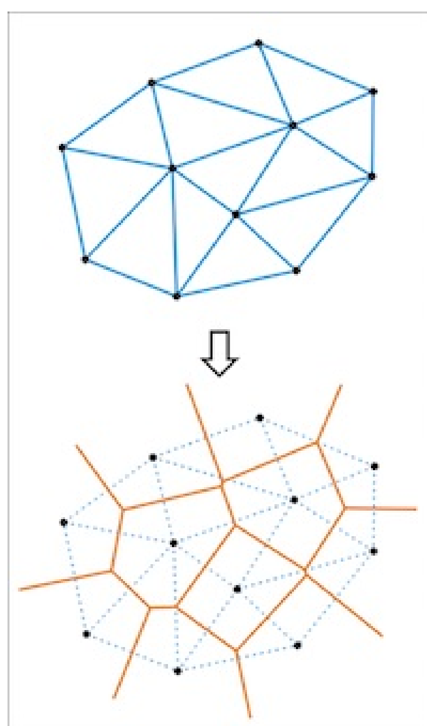

Figura 2.4: Relazione duale tra triangolazione di Delaunay (sopra) e tassellatura di Voronoi(sotto)

Quindi, un arco AB appartiene a una triangolazione di Delaunay se e solo se esiste un cerchio passante per A e B tale che preso ogni altro punto C appartenente all'insieme di vertici V, C non si trova all'interno di tale cerchio. In questo caso, si dice che AB è un *Arco di Delaunay*. Una triangolazione che possiede solo archi di Delaunay rispetta la *proprietà*

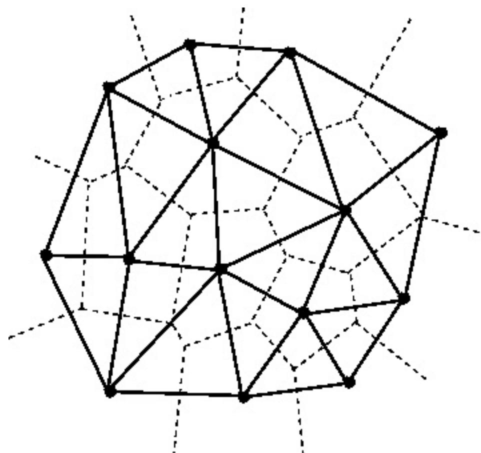

Figura 2.5: Triangolazione di Delaunay e Tassellatura di Voronoi

equiangolare locale.

Proprietà 2.1.3 (Proprietà equiangolare)**.** *Dato un insieme di* 4 *punti* $V = \{A, B, C, D\}$ *e la sua triangolazione di Delaunay* $DT(V)$*, supponiamo che* $DT(V)$ *sia composta da due triangoli* ABC *e* BCD*. Allora, la somma degli angoli in* A *e in* D *è minore di* 180*.*

Tale proprietà è molto importante perché è alla base degli algoritmi per la costruzione di una triangolazione di Delaunay. A questa bisogna aggiungere la seguente Osservazione:

Osservazione 1. La somma degli angoli interni di un quadrilatero è 360.

Utilizzando l'Osservazione 1, è facilmente intuibile che se la somma degli angoli in A e in D fosse maggiore di 180 allora la somma degli angoli in B e in C sarebbe minore di 180, quindi la triangolazione formata dai triangoli ABD e ACD sarebbe di Delaunay. Questa osservazione sta alla base del *flip algorithm*, il quale viene utilizzato per correggere una qualsiasi triangolazione T al fine di renderla di Delaunay.

Dato un insieme di punti V, la triangolazione di Delaunay non è sempre unica: nel caso in cui esista una circonferenza passante per 4 punti di V, la triangolazione non risulta univocamente determinata.

Definizione 2.7 (Cocircolarità)**.** Un insieme di punti V si dice *Cocircolare* se ogni punto in V risiede nella stessa circonferenza. In questo caso, si dice che l'insieme V gode della proprietà di *Cocircolarità*.

In genere viene fatta l'assunzione che i punti del piano per cui si vuole costruire la triangolazione di Delaunay siano disposti in *posizione generale*, cioè non siano cocircolari a 4 a 4, in modo da garantirne l'unicità.

Infine, una triangolazione di Delaunay gode della seguente proprietà:

Teorema 2.1.4. *In una triangolazione di Delaunay, il numero medio di vicini di un punto è uguale a 6.*

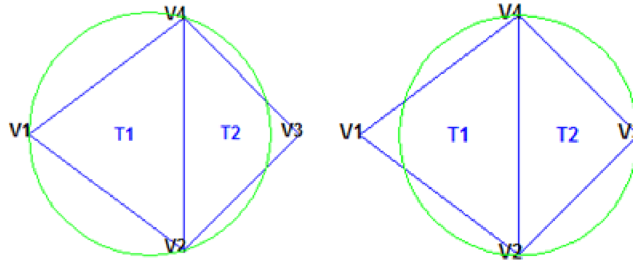

Figura 2.6: Triangolazione di Delaunay su V1, V2, V3 e V4

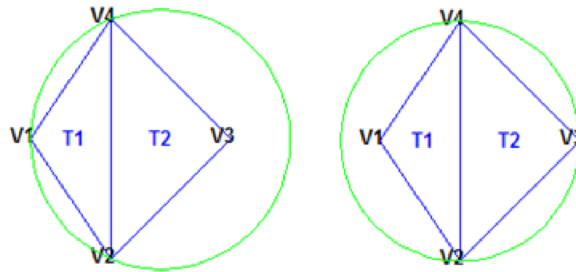

Figura 2.7: Normale triangolazione su V1, V2, V3 e V4

La Figura 2.6 mostra un esempio di corretta triangolazione di Delaunay: il cechio passante per i vertici V1-V2-V4 non contiene V3 e il cerchio passante per i vertici V2-V3-V4 non contiene V1. Invece, la Figura 2.7 mostra una triangolazione che non è di Delaunay: infatti, il cerchio passante per i vertici V1-V2-V4 contiene V3 e il cerchio passante per i vertici V2-V3-V4 contiene V1.

Curse of dimensionality. Prima della descrizione degli algoritmi per la costruzione della triangolazione di Delaunay proposti in letteratura è importante considerare il problema della *curse of dimensionality* [1]. La curse of dimensionality è un problema che affligge molte tecniche di ripartizione spaziali, e consiste in un aumento esponenziale della complessità al caso pessimo degli algoritmi in relazione all'aumento della dimensione dello spazio euclideo in cui i punti sono considerati. Sia la tassellatura di Voronoi sia la triangolazione di Delaunay soffrono di tale problema, che nel caso specifico della triangolazione di Delaunay equivale in un aumento esponenziale del numero medio di vicini per nodo all'aumentare della dimensione dello spazio.

2.2 Algoritmi per la costruzione della Triangolazione di Delaunay

In questa sezione si propone un'analisi dei principali algoritmi per la costruzione della triangolazioni di Delaunay. Esistono molti algoritmi in letteratura, che vanno da algoritmi incrementali, algoritmi divide et impera, ma anche algoritmi che banalmente costruiscono una triangolazione qualunque per poi *trasformarla* successivamente verificando le proprietà descritte nella sezione precedente.
Gli algoritmi che sono descritti in questa sezione sono il *Flip Algorithm* [9], che costruisce in modo incrementale la triangolazione andando ad effettuare un flip degli archi che non soddisfano la proprietà equiangolare, l'*algoritmo di Bowyer-Watson* [31], che costruisce la triangolazione in modo incrementale utilizzando un *supertriangolo* di supporto, e un algoritmo basato sulla tecnica del *Divide et Impera*.

2.2.1 Il Flip Algorithm

Il *Flip Algorithm* [9] è un *algoritmo incrementale* che si basa sulla Proprietà 2.1.3 descritta nella sezione precedente. Un algoritmo incrementale è un algoritmo che costruisce la soluzione al problema aumentando la dimensione del problema un passo alla volta, e non la calcola tutta insieme.
Fondamentale per la descrizione dell'algoritmo è il *test del cerchio vuoto.*

Test del Cerchio Vuoto

Si dice *Test del Cerchio Vuoto* il test che effettuiamo per verificare che la circonferenza passante per i 3 punti di un qualsiasi triangolo della triangolazione non contenga alcun punto del piano al suo interno. Si noti che per piani euclidei a dimensione $d > 2$ il cerchio è una ipersfera.
In un piano euclideo a due dimensioni, se i punti A, B, C sono i punti del triangolo nel quale si sta effettuando il test, e D è il punto non appartenente al triangolo, un metodo per effettuare il test è quello di valutare il determinante della matrice

$$\begin{vmatrix} X_A & Y_A & {X_A}^2 + {Y_A}^2 & 1 \\ X_B & Y_B & {X_B}^2 + {Y_B}^2 & 1 \\ X_C & Y_C & {X_C}^2 + {Y_C}^2 & 1 \\ X_D & Y_D & {X_D}^2 + {Y_D}^2 & 1 \end{vmatrix}$$

dove i punti A, B e C sono disposti in senso antiorario e X_i, Y_i rappresentano le ascisse e le ordinate dei diversi punti;

- se il determinante è maggiore di 0, allora D è incluso nel cerchio, quindi il cerchio non è vuoto e la proprietà non è verificata per il triangolo ABC.

- se il determinante è minore di 0, allora D è all'esterno del cerchio, quindi si passa a verificare per un altro punto del piano o, se D è l'ultimo punto, il test è verificato.

Esempio. Si consideri il triangolo ABC e il punto D descritti in Figura 2.8. Come si evince

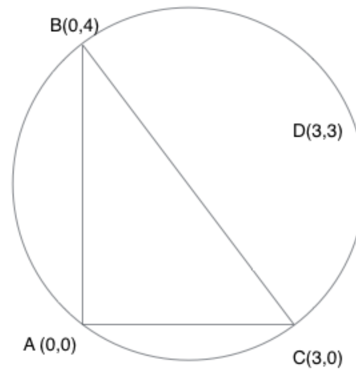

Figura 2.8: Triangolo ABC con punto D all'interno della circonferenza che lo circoscrive
.

dalla figura, il punto D si trova all'interno del cerchio. Consideriamo quindi i punti in ordine antiorario partendo da B (B, poi A , poi C e infine D). Si calcoli quindi il determinante della matrice

$$det\begin{vmatrix} 0 & 4 & 0^2+4^2 & 1 \\ 0 & 0 & 0^2+0^2 & 1 \\ 3 & 0 & 3^2+0^2 & 1 \\ 3 & 3 & 3^2+3^2 & 1 \end{vmatrix} = det\begin{vmatrix} 0 & 4 & 16 & 1 \\ 0 & 0 & 0 & 1 \\ 3 & 0 & 9 & 1 \\ 3 & 3 & 18 & 1 \end{vmatrix} = 36 > 0.$$

Si consideri adesso la configurazione dei punti in Figura 2.9. In questo caso, il punto D non si trova all'interno del cerchio, infatti effettuando i calcoli si ottiene:

$$det\begin{vmatrix} 0 & 4 & 0^2+4^2 & 1 \\ 0 & 0 & 0^2+0^2 & 1 \\ 3 & 0 & 3^2+0^2 & 1 \\ 5 & 3 & 5^2+3^2 & 1 \end{vmatrix} = det\begin{vmatrix} 0 & 4 & 16 & 1 \\ 0 & 0 & 0 & 1 \\ 3 & 0 & 9 & 1 \\ 5 & 3 & 34 & 1 \end{vmatrix} = -84 < 0.$$

il determinante è negativo.

Flip degli archi

Una volta effettuato il test del cerchio vuoto, si utilizza l'Osservazione 2.1.3 per trasformare la triangolazione in una triangolazione di Delaunay. Riprendendo gli esempi precedenti,

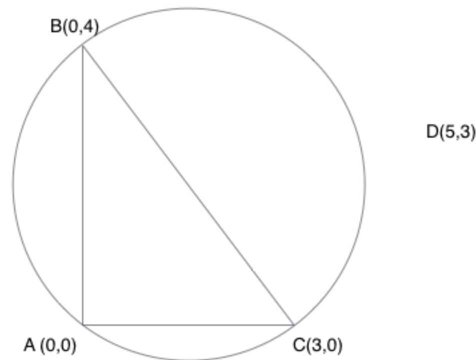

Figura 2.9: Triangolo ABC con punto D all'esterno della circonferenza che lo circoscrive

.

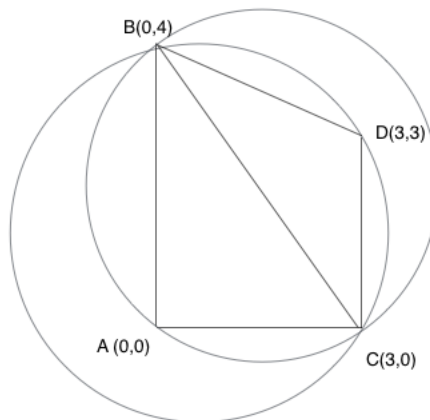

Figura 2.10: Triangolazione che non verifica la proprietà del cerchio vuoto

.

si consideri la triangolazione in Figura 2.10: si noti che sono mostrate le circonferenze che circoscrivono i triangoli della triangolazione. Questi triangoli non soddisfano la proprietà del cerchio vuoto, poiché il nodo D risiede all'interno del cerchio di ABC, mentre A risiede all'interno di quello del triangolo BDC. Per rendere tale triangolazione *di Delaunay*, eseguiamo il *flip* del lato BC. In questo modo, i nuovi triangoli della triangolazione diventano ACD e DBA. In questo modo (Figura 2.11), la triangolazione verifica la proprietà del cerchio vuoto.

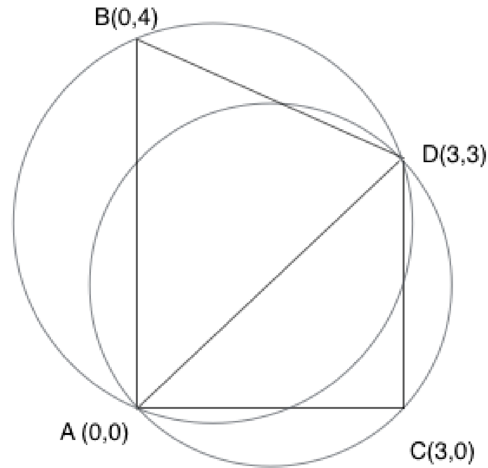

Figura 2.11: Triangolazione di Delaunay sui nodi A, B, C, D

.

Algoritmo

Nella descrizione dell'algoritmo, supponiamo che al tempo t siano presenti già n punti nel piano e che al tempo $t+1$ venga inserito un nuovo punto v_{n+1}. L'idea di base è quella di aggiungere il nuovo punto alla triangolazione di Delaunay già presente e, tramite il flipping degli archi, *trasformare* localmente i triangoli non di Delaunay al fine di mantenere corretta la triangolazione. L'Algoritmo 1 descrive il comportamento dell'algoritmo dopo l'inserimento

Algorithm 1 Flip Algorithm

1: **procedure** INSERTION(Vertex v)
2: $T_i \leftarrow$ triangle that contains v on current triangulation DT
3: add edges to link v to the points of T_i
4: **while** empty circle test fails for some triangle T on triangulation DT **do**
5: *flip wrong edge on* T

di un nuovo punto: in dettaglio, dopo l'inserimento di un punto v_{n+1}:

- viene trovato nella triangolazione il triangolo T_j che contiene il punto v_{n+1}.
- si divide T_j in 3 sottotriangoli T_{j_1}, T_{j_2} e T_{j_3}.

- viene effettuato il test del cerchio vuoto per ogni triangolo T_i della triangolazione, effettuando i flip dei lati quando il test non viene verificato.

Esempio. Il seguente esempio mostra l'esecuzione dei passi dell'algoritmo Flip Algorithm in relazione all'inserimento di un nuovo nodo n in una rete V. Si consideri la triangolazione di Delaunay relativa alla rete V prima dell'inserimento del nuovo nodo, mostrata in Figura 2.12.

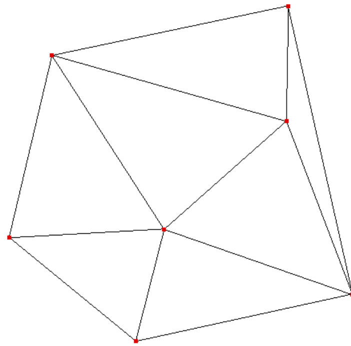

Figura 2.12: La triangolazione di Delaunay di V prima dell'inserimento di n

.

Durante la fase di join, il nodo n identifica il triangolo che lo contiene nella triangolazione, il quale viene diviso in tre sotto-triangoli T_1, T_2 e T_3 come mostrato in figura 2.13.

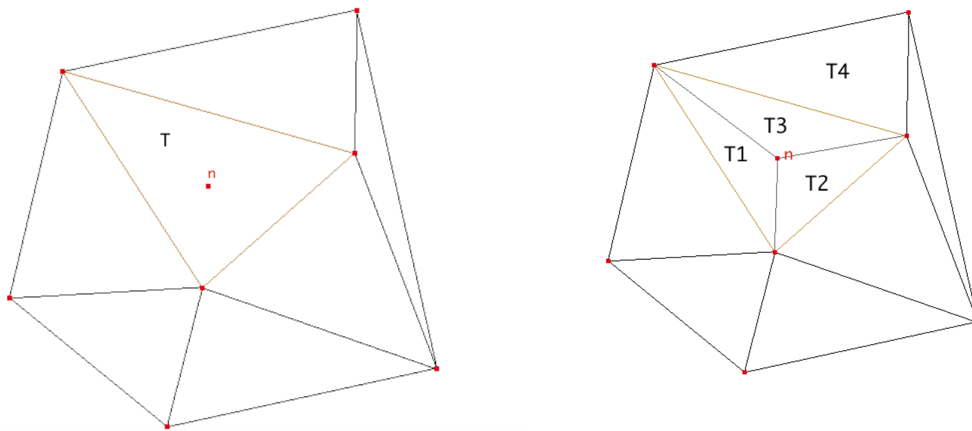

Figura 2.13: A sinistra: il nodo n si posiziona nella rete. A destra: il triangolo T viene suddiviso nei sottotriangoli T_1, T_2 e T_3

.

A questo punto, ognuno dei lati del triangolo T viene controllato, al fine di verificare che sia ancora un arco di Delaunay del grafo. Supponiamo che il primo lato a essere verificato sia quello relativo al nuovo triangolo T_3: viene quindi controllato se la circonferenza passante per i vertici del triangolo T_4 contiene qualche nodo di V. Poiché tale circonferenza contiene il nodo n, il lato effettua un'operazione di flip (Figura 2.14).
L'operazione di flip ha permesso la creazione nella triangolazione dei due nuovi triangoli T_5 e T_6, mentre T_4 e T_3 sono stati eliminati. Viene quindi effettuato il test del cerchio vuoto sul triangolo T_7, il quale possiede un lato in comune con il nuovo triangolo T_5 (Figura 2.14 a destra).

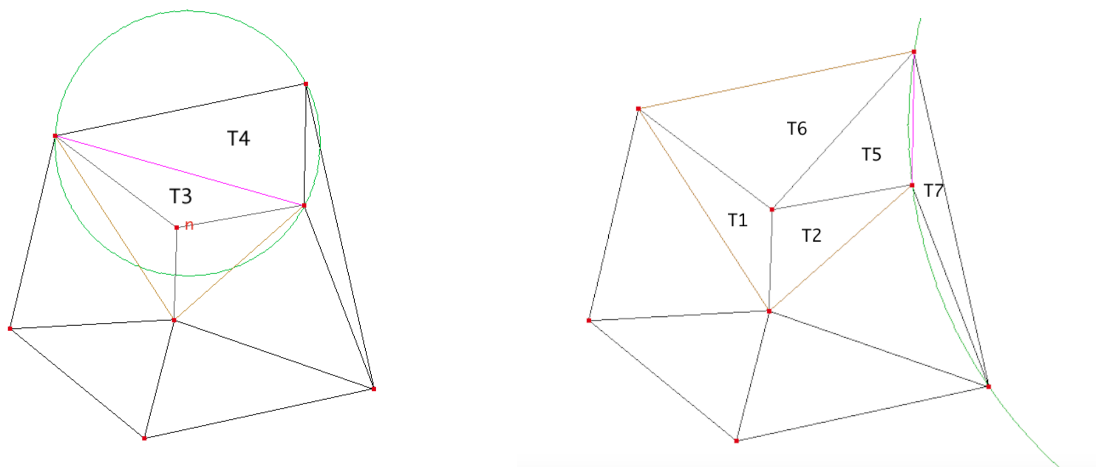

Figura 2.14: A sinistra: il circumcerchio del triangolo $T4$ contiene n. A destra: Test del cerchio vuoto sul nuovo triangolo T_5
.

Questa volta, il test è verificato, quindi non ci sono operazioni di flip di lati. L'algoritmo continua nella verifica dei lati relativi al triangolo originale T. In dettaglio, la figura 2.15 mostra il test del cerchio vuoto applicato al triangolo T_8, che non è verificato. Quindi, il lato originale di T flippa, come mostrato in Figura 2.16.

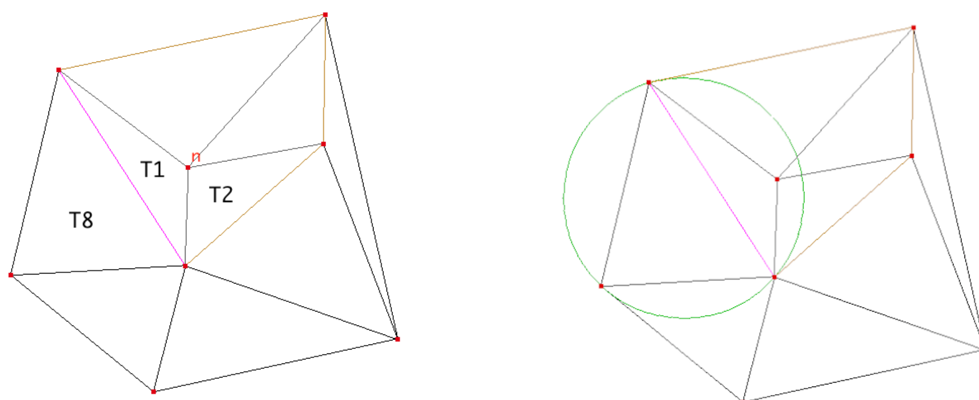

Figura 2.15: Teste del cerchio vuoto applicato al triangolo T_8

.

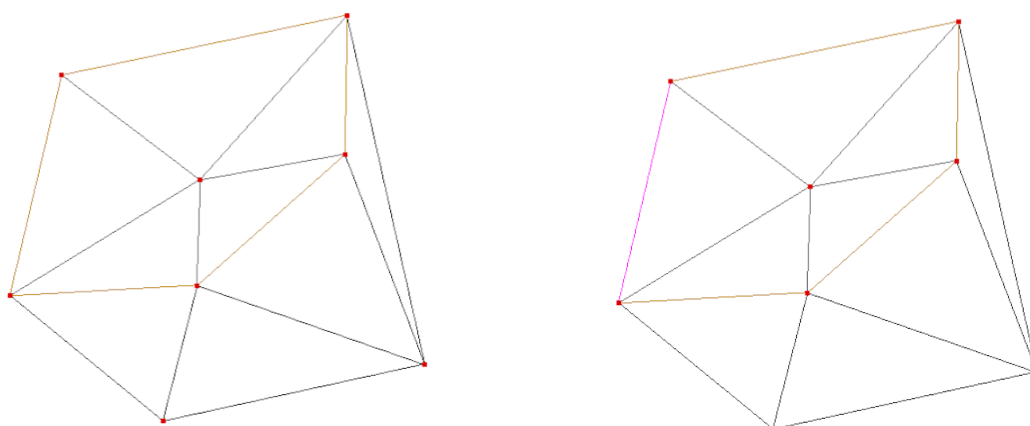

Figura 2.16: Test del cerchio vuoto applicato al triangolo T_8

.

L'algoritmo procede ricorsivamente controllando tutti i triangoli della triangolazione, finché la triangolazione ottenuta è la nuova triangolazione di Delaunay con il nodo n correttamente inserito. I prossimi passi di questo sono mostrati in Figura 2.17.

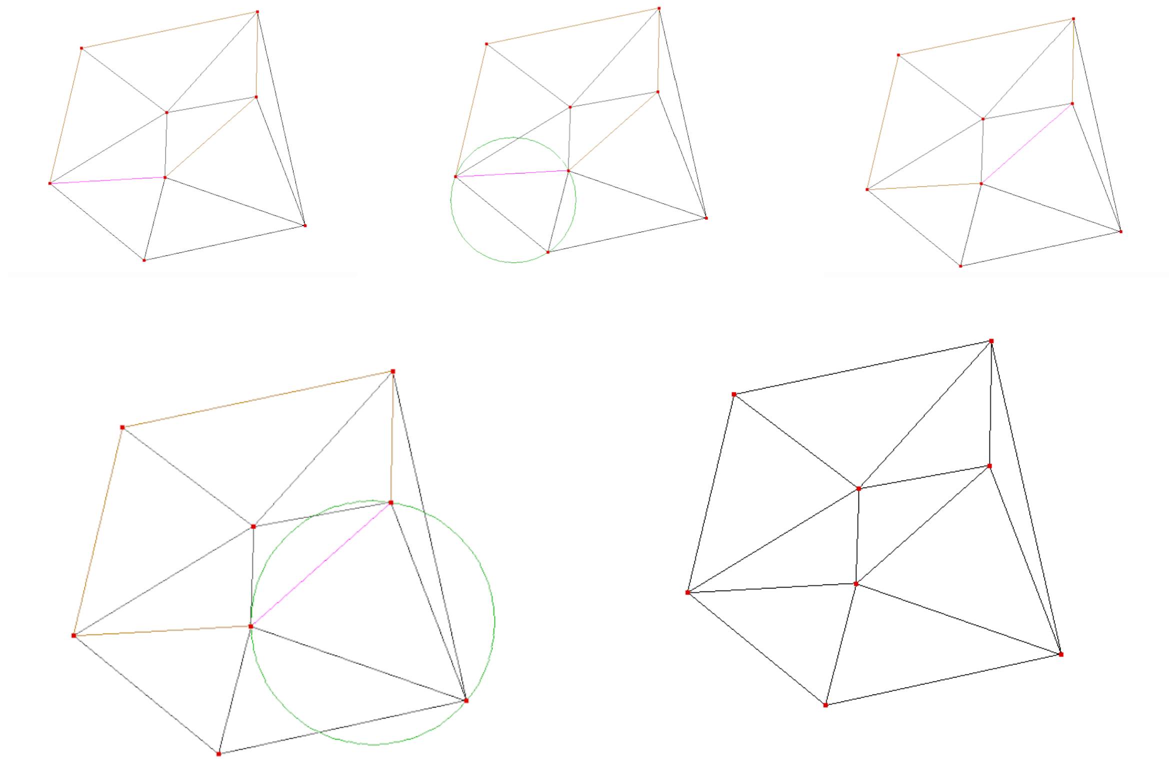

Figura 2.17: Ultimi passi dell'algoritmo Flip Algorithm

.

2.2.2 Algoritmo di Bowyer-Watson

Come il *Flip Algorithm* [9], anche l'*Algoritmo di Bowyer-Watson* [31] è un algoritmo incrementale. L'idea di base dell'algoritmo è che l'aggiunta di un nuovo punto all'insieme dei punti V su cui è definita una triangolazione di Delaunay ha un effetto locale su di essa.
Al momento dell'inserimento di un nuovo punto v_{n+1} nel piano, viene effettuata una ricerca, nella triangolazione già calcolata DT, dei triangoli T_i tali che il circumcerchio di questi contenga v_{n+1}: questi vengono catalogati come *BadTriangles*. Sia $\bigcirc_{T_i}$ il circumcerchio relativo al triangolo T_i: allora, l'insieme dei BadTriangles è definito formalmente come:

$$BadTriangles = \{T_i \in DT | \exists v_k \in V \ such\ that\ v_k \in \bigcirc_{T_i}\} \tag{2.1}$$

Successivamente, tali triangoli vengono rimossi dalla triangolazione, creando un poligono non triangolarizzato in questa, che viene definito come *Star Shaped Polygon* (Figura 2.18).

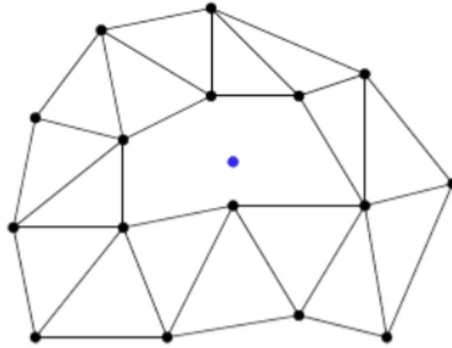

Figura 2.18: il poligono Star Shaped Polygon

.

Quindi, lo star shaped polygon viene triangolato nuovamente ottenendo una triangolazione di Delaunay corretta nell'insieme dei punti $V \cup \{v_{n+1}\}$.
L'algoritmo può essere utilizzato per la costruzione della triangolazione di Delaunay di un insieme di punti V. Prima di procedere all'inserimento del primo punto, si aggiungono 3 punti all'insieme V tali che il triangolo S di cui questi tre punti sono i vertici sia abbastanza grande da contenere tutti i punti di V. Tale triangolo prende il nome di *supertriangolo*. L'utilizzo del supertriangolo è necessario per la costruzione di una triangolazione corretta perché se il punto v_k da aggiungere al piano si trova all'esterno del convex-hull boundary della triangolazione DT, allora non è possibile identificare lo shaped star polygon su cui eseguire la nuova triangolazione.
Una volta inseriti tutti i punti di V, il supertriangolo e tutti gli archi della triangolazione connessi a uno dei suoi vertici sono eliminati. L'Algoritmo 2 mostra la procedura in dettaglio.

Esempio. Si considerino i punti A, B, C, D in Figura 2.19. Il primo passo dell'algoritmo è la creazione del supertriangolo, il quale viene aggiunto all'insieme dei triangoli della triangolazione, come mostrato in Figura 2.19.

Algorithm 2 Algoritmo di Bowyer-Watson

```
1: procedure BOWYER-WATSON(NodeList nl)
2:     Triangulation ← ∅
3:     add Supertriangle to Triangulation including
4:     for Node n in nl do
5:         BadTriangles ← ∅
6:         for Triangle T in Triangulation do
7:             if n is inside circumcicle of T then
8:                 add T to BadTriangles
9:         starShapedPolygon ← ∅
10:        for Triangle T in BadTriangles do
11:            for Edge e in T do
12:                if e is not shared by any another triangle in BadTriangles then
13:                    add e to starShapedPolygon
14:        for Triangle T in BadTriangles do
15:            remove T from Triangulation
16:        for Edge e in starShapedPolygon do
17:            create new Delaunay Triangle T_new with n and e
18:            add T_new to Triangulation
19:    for Triangle T in Triangulation do
20:        remove T if contains at least one node of SuperTriangle
       return Triangulation
```

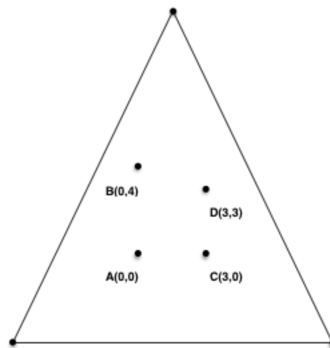

Figura 2.19: In Figura: il supertriangolo include i punti A, B, C, D

.

Quindi viene eseguito un ciclo per l'aggiunta dei punti dell'insieme. Si supponga che i punti vengano aggiunti al piano seguendo l'ordine alfabetico. Il primo punto da aggiungere è A, il quale è contenuto dal supertriangolo, che quindi viene collegato ai suoi vertici e si forma cosi il triangolo $T1$, come mostrato in Figura 2.20.

Viene quindi aggiunto il punto B, e viene quindi calcolato l'insieme $BadTriangles$. A questo punto dell'esecuzione, i triangoli $T1$ e $T4$ sono inclusi in $BadTriangles$. (Figura 2.20).

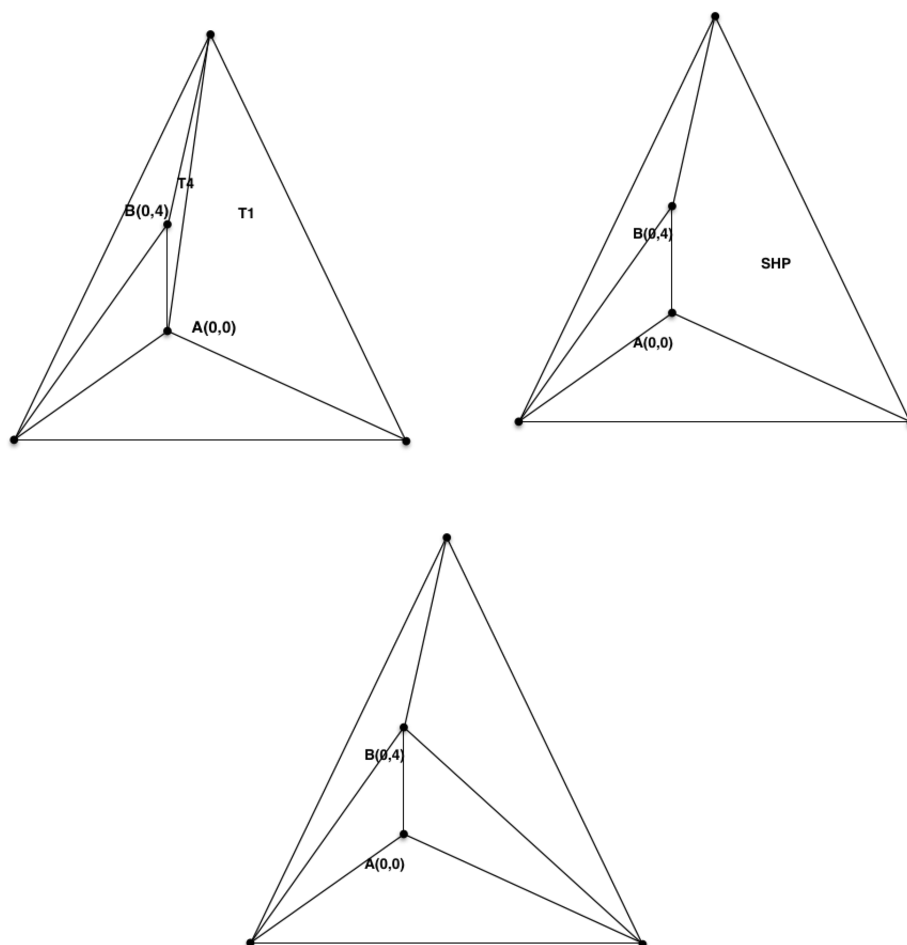

Figura 2.20: Passi dell'algoritmo di Bowyer-Watson

.

La Figura 2.20 mostra i passi di esecuzione dell'algoritmo descritto nell'esempio considerato: in alto a sinistra mostra i triangoli $T1$ e $T4$ sono inseriti in $BadTriangles$ dopo l'aggiunta del punto B. In alto a destra è invece mostrato lo star shaped polygon (indicato con SHP) risultante dall'eliminazione dei $BadTriangles$. Sempre in Figura 2.20, in basso è mostrata la triangolazione risultante dopo l'esecuzione della fase di triangolazione dello star shaped polygon.

I triangoli in $BadTriangles$ sono quindi eliminati, creando lo star shaped polygon. Viene quindi costruita la triangolazione di Delaunay di tale poligono). In seguito, si opera ricorsiva-

mente, secondo la solita procedura, per ogni punto che viene aggiunto. Infine, i vertici e tutti gli archi della triangolazione collegati con il supertriangolo sono rimossi, e la triangolazione ottenuta è di Delaunay.

2.2.3 Algoritmi Divide et Impera

Gli algoritmi *Divide et Impera* per il calcolo della triangolazione di Delaunay risultano essere il modo più efficiente per la sua costruzione [37]. La maggior parte di questi algoritmi opera in due dimensioni, anche se è stato descritto un algoritmo per d dimensioni, con $d > 2$, in [11]
L'algoritmo Divide et Impera presentato in questa sezione è quello sviluppato da Guibas e Stolfi [15].
L'idea base è la seguente: viene suddiviso il piano euclideo da sinistra verso destra in modo da avere i nodi raggruppati in insiemi di 2 o 3 elementi. In seguito, si costruisce la triangolazione di Delaunay di questi insiemi e infine si fondono le triangolazioni calcolate a due a due, finché non ricostruiamo l'insieme di nodi iniziale.

Algoritmo

Dato un insieme di punti $V = \{v_1, v_2, \ldots, v_n\}$ in un piano euclideo P_E, l'algoritmo segue i seguenti punti:

1. si divide V in *sottoinsiemi* di due o tre elementi V_i, partendo da sinistra verso destra, dall'alto verso il basso. In questo modo, è assegnato un *ordinamento lessicografico* ai punti di V.

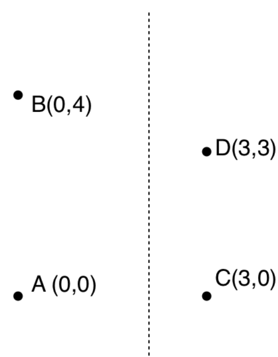

Figura 2.21: Primo passo dell'algoritmo *Divide et Impera* di Guibas e Stolfi.

2. si costruiscono le triangolazioni di Delaunay degli insiemi V_i.

3. Si fondono i V_i a due a due, finché non rimane un unico insieme. In fase di fusione, distinguiamo i seguenti tipi di archi tra i punti di P_E:

   - gli archi LL, che si sono stati creati nel gruppo di sinistra.
   - gli archi RR, che sono stati creati nel gruppo di destra.
   - gli archi LR, che si formeranno quando saranno fusi gli insiemi.

   Nonostante sia possibile che vengano eliminati della triangolazione finale archi LL o RR, è certo che non ne sono creati di nuovi.

   Inizialmente, viene inizializzato il Base LR, definito come il più *basso* (in termini di coordinate) arco che collega i nodi dell'insieme di sinistra S con l'insieme di destra D.

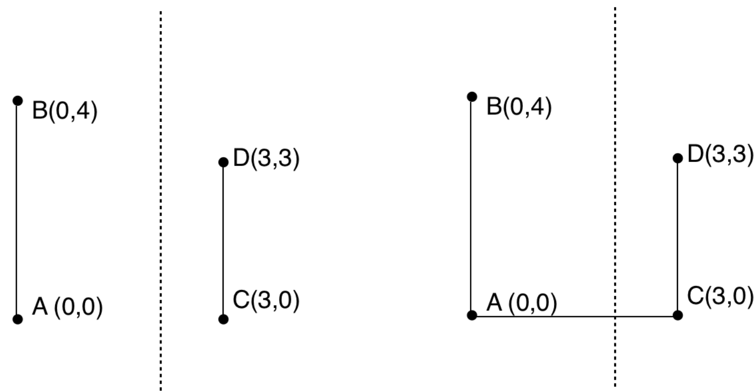

Figura 2.22: Secondo e terzo passo dell'algoritmo *Divide et Impera* di Guibas e Stolfi. Viene aggiunto l'arco Base LR tra A e C

.

Viene quindi scelto un punto *candidato* per l'endpoint destro e uno per quello sinistro. Inizialmente, procediamo con il candidato destro: il prossimo candidato destro è il nodo collegato all'endpoint destro del Base LR che con lui ha l'angolo minore in senso orario. Viene quindi controllato:

(a) se tale angolo è minore di 180 gradi (*prima condizione*);

(b) se il *prossimo candidato* non sta nel circumcerchio formato dagli endpoint dell'arco Base LR e il candidato attuale (*seconda condizione*).

Distinguiamo i seguenti casi:

- vale la prima condizione e la seconda condizione: il nodo candidato viene scelto come endpoint del nuovo LR.

- non vale la prima condizione: nessun candidato per il lato viene scelto.
- vale la prima condizione ma non vale la seconda: l'arco RR che collega il candidato con l'endpoint del Base LR destro viene eliminato.

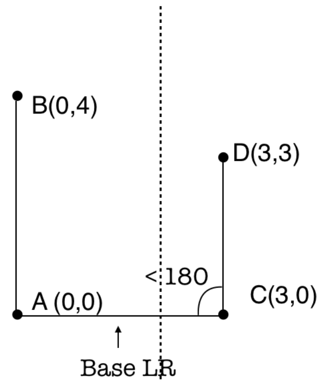

Figura 2.23: Quarto passo dell'algoritmo Divide et Impera

.

La Figura 2.23 mostra il quarto passo dell'algoritmo Divide et Impera: Il nodo D è il candidato destro scelto, poiché l'angolo in figura è inferiore a 180 e il prossimo candidato destro (che non esiste) non sta all'interno del circumcerchio di A,C e D.

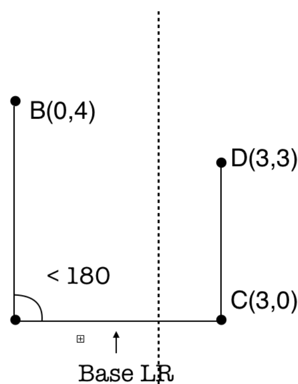

Figura 2.24: Quinto passo dell'algoritmo Divide et Impera

.

La Figura 2.24 mostra il quinto passo dell'algoritmo Divide et Impera: il nodo B è il candidato sinistro scelto, poiché l'angolo in figura è inferiore a 180 e il prossimo candidato sinistro (che non esiste) non sta all'interno del circumcerchio di A,C e B.

Se nessun candidato viene selezionato per S e D, allora la fusione è completa, altrimenti se ne viene selezionato soltanto uno, si collega quel punto all'endpoint (destro o sinistro

dipende dal candidato scelto) del Base LR e l'arco appena formato diventa il nuovo arco Base LR. Se sia il candidato destro che il candidato sinistro vengono scelti, viene effettuato il seguente test:

- se il candidato destro sta all'interno del circumcerchio definito dagli endpoints dell'arco Base LR e il candidato sinistro, allora viene scelto il candidato sinistro,
- altrimenti viene scelto il candidato destro.

É garantito che questa condizione valga solamente per uno dei due candidati se e solo se non esistono nella rete insiemi di nodi cocircolari.

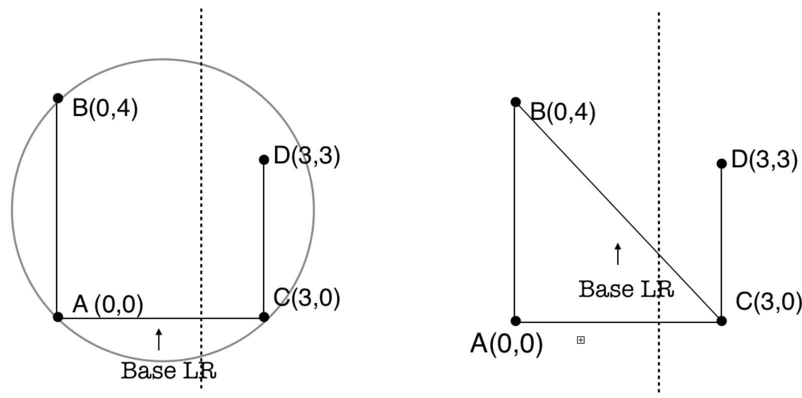

Figura 2.25: Scelta del nodo B come candidato

.

In Figura 2.25 viene mostrata la scelta del nodo B come nodo candidato scelto: infatti, il nodo D sta all'interno del circumcerchio del triangolo di cui A,B e C sono i vertici.

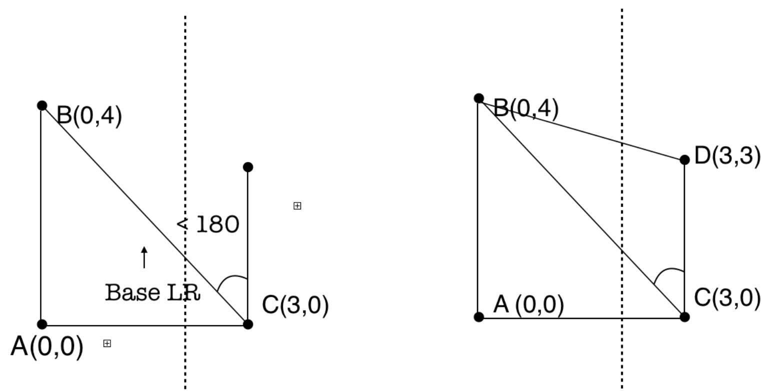

Figura 2.26: *Algoritmo Divide et Impera.* Passi finali dell'algoritmo sui nodi A, B, C, D

.

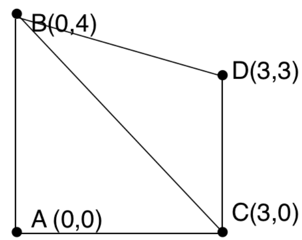

Figura 2.27: *Algoritmo Divide et Impera.* Triangolazione di Delaunay calcolata dall'algoritmo
.

La Figura 2.27 mostra infine la triangolazione di Delaunay finale ottenuta utilizzando l'algoritmo.

Capitolo 3

Algoritmi distribuiti per Delaunay overlay

In questo capitolo è presentata una rassegna dei principali algoritmi distribuiti per la costruzione della triangolazione di Delaunay. Le triangolazioni di Delaunay sono state recentemente utilizzate soprattutto per la loro applicazione nel campo delle reti di sensori e delle reti ad-hoc.
In ambiente distribuito, i principali campi di applicazione della triangolazione di Delaunay sono i seguenti:

- applicazioni che utilizzano routing geografico;
- ambienti virtuali distribuiti (DVE);
- reti distribuite e P2P;
- area coverage.

Le triangolazioni di Delaunay posssono essere utilizzate per modellare problemi di routing geografico in un insieme di nodi, poiché se so considera il grafo $G(V, E)$ dove l'insieme dei nodi coincide con i punti nel piano euclideo e l'insieme dei collegamenti tra i nodi coincide con gli archi della triangolazione di Delaunay, è garantita la terminazione di algoritmi greedy. Questo significa che è possibile utilizzare algoritmi di semplice implementazione garantendo un risultato ottimale. Oltre ad algoritmi greedy per il routing, le triangolazioni di Delaunay supportano il *compass routing* [20]. Invece di determinare il nodo a cui inviare il messaggio minimizzando la distanza euclidea dalla destinazione, il compass routing utilizza gli ancoli per la scelta del nodo a cui inoltrare il messaggio. In dettaglio, siano $s \in V$ il nodo sorgente del messaggio m e $d \in V$ il nodo destinatario, e supponiamo che i vicini di s nel grafo siano denotati con $s_1, \ldots, s_j$. Nella procedura di scelta del nodo a cui inviare m, s considera gli angoli α_i formati dall'intersezione della retta passante per s e per ogni s_i, vicino di s, e della

retta passante per s e d, con $i = 1, \ldots, j$. Quindi, s invia il messaggio m al vicino relativo all'angolo α_i minore.
Esempi di applicazioni di routing geografico sono:

- query effettuate da un sistema centralizzato attraverso una rete di sensori per avere informazioni su una determinata area;
- query effettuate da sensori verso un sistema centralizzato per la segnalazione del verificarsi di un evento nell'area geografica.

La sezione 5.1 presenta alcuni algoritmi orientati al routing geografico: GeoPeer, sviluppato da F. Araùjo e L.Rodrigues [3], che consiste in una triangolazione di Delaunay aumentata con archi a lungo raggio, e Skip Delaunay Network, di Tsuboi et al. [38], che risolve il problema del grande diametro della rete costruendo un struttura a livelli di triangolazioni di Delaunay.

Nella sezione 5.2 sono presentati alcuni algoritmi che prendono in considerazione che non sempre i sensori hanno un raggio di trasmissione tale da poter comunicare con tutti i propri vicini. Tali algoritmi, chiamati *algoritmi di localizzati*, eseguono la scelta del passo da eseguire basandosi solamente sull'informazione locale. Algoritmi localizzati già presentati sono, per esempio, l'algoritmo greedy routing e l'algoritmo compass routing, poiché guardano alla conoscenza locale per decidere a chi inoltrare il messaggio.
La sezione 5.2 presenta alcuni algoritmi localizzati per la costruzione di strutture basate su triangolazione di Delaunay che garantiscano il supporto al greedy e al compass routing. In particolare, una di queste strutture è chiamata *Planar Localized Delaunay Triangulation* (PLDel), e garantisce che un cammino tra due nodi u e v non sia più lungo di t volte il cammino minimo possibile nel grafo originale centralizzato. Gli algoritmi che la costruiscono sono proposti da Xiang-Yang Li et al. [23], e da Araùjo et al. [2], i quali realizzano una procedura per ottenere lo stesso grafo utilizzando un basso costo di comunicazione. In sezione 5.2 è analizzato inoltre un algoritmo per costruire una struttura con proprietà simili a PLDel chiamata Restricted Delaunay Network, di J. Gao et al. [12].

La sezione 5.3 descrive due algoritmi che utilizzano le triangolazioni di Delaunay per il supporto agli ambienti virtuali distribuiti, principalmente per la sincronizzazione delle *Aree di Interesse* (*AOI*) dei nodi: tali ambienti sono molto attuali, soprattutto nel capo videoludico, poiché permettono la costruzione e la gestione di interi mondi virtuali senza il bisogno di un supporto centralizzato. A causa del vincolo di consistenza della visione che i nodi devono avere all'interno del mondo virtuale (se un personaggio davanti a me compie un'azione, io devo essere in grado di vederla in tempo reale), è molto importante che sia garantito che il numero di messaggi in tali reti sia limitato. A tale scopo, si utilizzano le triangolazioni di

Delaunay, e in particolare la tassellatura di Voronoi, per identificare le aree di interesse di un nodo n, cioè quelle aree che al momento t devono essere aggiornate per tale nodo.
Algoritmi distribuiti tra gli analizzati nella sezione 5.3 applicati a questo campo sono l'algoritmo di E. Buyukkaya et al. [8] e l'algoritmo di Ghaffari et al. [14].

La sezione 5.4 descrive un'analisi di algoritmi generici per la costruzione della triangolazione di Delaunay distribuita utilizzati principalmente in reti P2P e di sensori: l'algoritmo di Liebeherr et al. [24], l'algoritmo di M. Steiner et al. [33].

Un problema di *area coverage* consiste nel trovare un posizionamento ottimo per un insieme di sensori in uno spazio geografico P in modo che tale spazio sia interamente coperto dai raggi dei sensori. Tale posizionamento, in genere, ha lo scopo di monitorare aree geografiche, soprattutto per motivazioni ambientali.
La sezione 5.5 prevede la descrizione di due algoritmi che possono essere utilizzati per tale applicazione: DT-Score, realizzato da Wu et al. [43], e l'algoritmo di Vu et al: [40]. Entrambi si pongono il problema dell'area coverage, ma i primi ottimizzano il posizionamento dei sensori utilizzando la proprietà del cerchio vuoto della triangolazione di Delaunay, mentre gli altri cercano di minimizzare le aree non coperte allargando in modo ottimale il raggio dei sensori nelle vicinanze di tali aree.

Infine, alcune applicazioni delle triangolazioni di Delaunay prevedono un loro utilizzo in campo grafico, per la creazione dei poligoni nelle mesh grafiche, e nel campo dell'apprendimento automatico, in particolare nell'algoritmo del *k-Nearest Neighbor* [17]. L'approfondimento di questi campi non rientra nello scopo della tesi, quindi si rimanda alla letteratura.

3.1 Algoritmi orientati al routing geografico

In questa sezione sono presentati alcuni algoritmi distribuiti per la costruzione della triangolazione di Delaunay orientati al routing geografico. La triangolazione di Delaunay si presta bene ad applicazioni relative al routing geografico per il suo supporto ad algoritmi di routing di semplice implementazione e grande efficacia, quali il greedy routing e il compass routing.
Purtroppo, questa struttura geometrica soffre del problema di avere un diametro di rete non limitato, il quale rappresenta un problema in molte applicazioni data l'esigenza di un numero basso di messaggi nella rete. Per questo motivo, sono state introdotte delle soluzioni ispirate da alcuni overlay P2P al fine di ridurre tale diametro di rete.
Gli algoritmi presentati sono *GeoPeer* [3], il quale risolve il problema aggiungendo alla triangolazione di Delaunay un insieme di archi a lungo raggio, e *Skip Delaunay Network* [38], che crea una gerarchia di triangolazioni di Delaunay per i nodi della rete al fine di ridurre le distanze tra i nodi.

3.1.1 GeoPeer

GeoPeer è un sistema sviluppato presso l'*Università di Lisbona* da F. Araùjo e L.Rodrigues [3] con l'obiettivo di supportare queries geografiche efficienti. Esempi di tali servizi possono essere queries per risorse all'interno di territori geografici, oppure queries mirate a collezionare informazioni da sensori in ambito ambientale.
GeoPeer è implementato come un'overlay network P2P che si basa su gli indirizzi IP. I nodi di GeoPeer costruiscono una triangolazione di Delaunay *aumentata* con archi a lungo raggio (LRC). Araùjo et al. si pongono il problema della realizzazione di un'overlay network in grado di supportare problemi di routing geografico con la caratterizzazione di un piccolo *diametro* di rete.

Definizione 3.1 (Diametro di una rete)**.** Si dice *diametro* di una rete di nodi V la massima distanza tra ogni coppia di nodi in V, dove la distanza tra due nodi è definita come il minimum shortest path tra i due nodi.

In [38] vengono messi a confronto i principali overlay network utilizzati in reti P2P, tra cui Pastry [32], Tapestry [45], Chord [34], Koorde [19], i quali non sono in grado di risolvere efficientemente problemi geografici poiché l'indirizzo logico assegnato ai nodi della rete è monodimensionale e non adatto a rappresentare coordinate a due o più dimensioni. Al contrario, overlay costruiti con CAN [30], TOPLUS[13] e triangolazioni di Delaunay risolvono efficientemente problemi di routing geografico, ma non limitano in nessun modo il diametro di rete. In tal senso, un messaggio su una rete basata su triangolazione di Delaunay potrebbe dover effettuare molti hop prima di giungere a destinazione. Per risolvere questo problema, sulla base di quanto fatto per *eCan*[44], Araùjo et al. estendono la triangolazione di Delaunay classica con *archi a lungo raggio* (LRC). GeoPeer consiste in:

1. un algoritmo che crea la triangolazione di Delaunay distribuita, con meccanismi generali che mantengono la triangolazione in caso di join e leave dei nodi;

2. un algoritmo che si assicura che ogni chiave possibile sia posseduta da un solo nodo della rete. Tale nodo è il responsabile dell'area di cui fa parte quella chiave;

3. un algoritmo che esegue un routing all'interno dell'overlay network. Per semplicità e efficienza, visto che la rete è una triangolazione di Delaunay, l'algoritmo di routing utilizzato è il greedy routing algorithm;

4. un insieme di meccanismi per stabilire i LRC tra i nodi della rete.

Creazione e Mantenimento della triangolazione

Per la creazione e il mantenimento della triangolazione di Delaunay, i nodi scambiano periodicamente i messaggi con i loro vicini. Sono utilizzati i seguenti messaggi:

- il messaggio BEACON, usato da un nodo v per informare i propri vicini che v è ancora attivo nella rete;
- il messaggio JOIN, usato da un nodo n durante la sua fase di join nella rete;
- il messaggio FAILURE, usato per informare i nodi della rete che un determinato nodo f ha subito un fallimento;
- il messaggio TRIANGULATE, usato da un nodo n per proporre la propria configurazione di triangoli di Delaunay ai suoi vicini;
- il messaggio BREAKLINKS, usato per riconfigurare la rete dopo la fase di join e di leave.

L'algoritmo è interamente decentralizzato e consiste di tre step logici:

1. il nodo v_i effettua la fase di join nella rete. Per questa operazione, v_i contatta un qualsiasi nodo già partecipante alla rete chiamato p. p invia un messaggio JOIN per conto di v_i destinato a v_i: visto che v_i non è ancora entrato nella rete, il messaggio giungerà a un nodo v_j, il quale rappresenta il nodo più vicino a v_i. Il nodo v_j provvede quindi a inviare il messaggio JOIN a tutti i nodi che sa essere possibili vicini di v_i. Infine, v_j invia un messaggio JOIN di risposta a v_i contenente l'insieme di vicini di v_i contattati.
2. manutenzione dei nodi vicini: periodicamente, ogni nodo della rete invia dei messaggi BEACON ai suoi vicini in modo da notificarli della loro presenza attiva nella rete.
3. il calcolo della triangolazione di Delaunay: ogni nodo v calcola una triangolazione di Delaunay utilizzando la sua conoscenza locale della rete. In questo caso, v potrebbe trovare dei triangoli a cui appartiene: se ciò avviene, allora v invia un messaggio di TRIANGULATE a tutti i vicini che appartengono a tali triangoli. Quando un nodo w riceve un messaggio di TRIANGULATE da un nodo v, viene controllato se anche nella vista locale di w è presente il triangolo. Distinguiamo quindi due casi:

   - w include il triangolo nella sua vista, quindi invia a v un messaggio di TRIANGULATE.
   - w non include il triangolo nella sua vista, quindi invia a v un messaggio di BREAKLINKS.

   Se esiste un consenso tra tutti i nodi sulla triangolazione calcolata da v, allora la triangolazione di Delaunay locale di v è corretta e la procedura si ferma, altrimenti v utilizza le informazioni presenti nei messaggio di BREAKLINKS per correggere la propria triangolazione locale.

L'algoritmo è predisposto anche alla disconnessione volontaria e involontaria dei nodi dalla rete, poiché il secondo step permette di identificare in modo automatico quei nodi che non sono più attivi nella rete. Quando un nodo w si accorge di un suo vicino v che ha subito un fallimento, ricalcola la triangolazione e invia un messaggio FAILURE a tutti i suoi vicini. Ognuno di questi provvede, quindi, all'inoltro del messaggio FAILURE a tutti i propri vicini, in modo da permettere che tutti i nodi della rete vengano a conoscenza del fallimento di v. L'algoritmo considera anche reti in cui i nodi non verificano l'assunzione della *posizione generale*, cioè in cui i nodi possono essere cocircolari a 4 a 4 : si ricorda che in quel caso la triangolazione di Delaunay non è più unica: in questo caso, se c_1 è cocircolare con altri nodi $c_2, \ldots, c_n, n > 3$, egli considera tali nodi tutti suoi vicini nella rete. In questo modo, c_1 diventa responsabile di loro e si mette in ascolto di loro probabili fallimenti.

Per quanto riguarda la consistenza della triangolazione di Delaunay distribuita calcolata dai nodi cocircolari, questi si sincronizzano utilizzando messaggi TRIANGULATE. Se uno dei nodi nella cocircolarità non è d'acccordo con la triangolazione costruita, questo invia un messaggio BREAKLINKS contenente tutti i nodi nella cocircolarità.

Divisione dello spazio

Il secondo compito di GeoPeer è quello di dividere lo spazio in modo da garantire che ogni punto in esso abbia uno e un solo responsabile tra i nodi della rete e che sia supportato il greedy routing per queries geografiche. Come descritto nella sezione precedente, grazie al supporto al greedy routing e al compass routing, la triangolazione di Delaunay garantisce il supporto alle queries geografiche. Per la suddivisione dello spazio, invece, non è possibile utilizzare le celle della tassellatura Voronoi di cui i nodi sono i vertici, poiché tali celle possono avere intersezioni non vuole con i triangoli della triangolazione di Delaunay, quindi non garantendo che tutti i punti del piano abbiano uno e un solo responsabile (Figura 3.1).

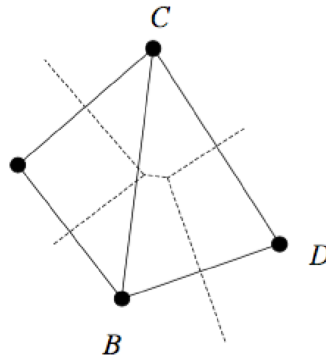

Figura 3.1: Le celle di Voronoi dei nodi intersecano la triangolazione di Delaunay

Non essendo la tassellatura di Voronoi, la suddivisione del piano utilizzata da GeoPeer è la seguente: si traccia la circonferenza che circoscrive ogni triangolo della triangolazione e si collega il centro di tale circonferenza con le bisettrici del triangolo. Se il triangolo è ben formato, allora tale centro si trova all'interno del triangolo e la procedura descritta divide il triangolo in tre parti: A_1, A_2 e A_3, determinando le aree di interesse dei punti (Figura 3.2). Se il triangolo non è ben formato, il centro è fuori dal triangolo ma la divisione è sempre possibile.

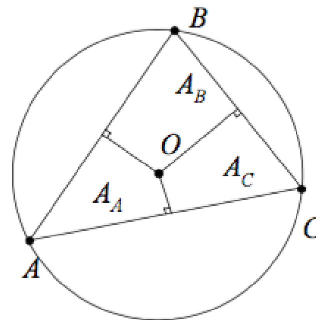

Figura 3.2: Le celle di Voronoi dei nodi intersecano la triangolazione di Delaunay

Per quanto riguarda le aree sul confine della triangolazione di Delaunay, li non è possibile applicare tale algoritmo, quindi si calcola il punto di interesse dei nodi sul *Convex Hull* semplicemente in base alla prossimità delle aree a tali nodi.

Algoritmo di routing

Il routing in GeoPeer si riconduce al routing in una triangolazione di Delaunay. È possibile utilizzare diversi algoritmi di routing in una triangolazione di Delaunay, e in particolare il più semplice è il greedy routing, come è stato già descritto nelle precedenti sezioni.
Araùjo e Rodrigues [3] descrivono nel loro paper una serie di algoritmi di routing, quali il *compass routing*, il *voronoi routing algorithm* e il *parallel voronoi*, ma decidono di utilizzare il greedy routing a causa della sua facile implementazione e della sua efficienza in campo pratico.

Meccanismi per i LRC

GeoPeer [3] include 4 meccanismi per la costruzione e il mantenimento di archi a lunga distanza LRC, col fine ultimo di ridurre il diametro della rete nella triangolazione di Delaunay costruita:

1. *Hop Level Mechanism.* Ogni volta che, nella consegna di un messaggio da parte di v_1 a v_n, vengono eseguiti b hop, viene tracciato un nuovo LRC tra v_1 e v_n. L'*Hop Level Mechanism* prevede anche la creazione di LRC di diverso livello: per esempio, v_1 dopo b hop giunge a v_2 e crea un LRC di livello 1 con v_2, v_2 dopo b hop giunge a v_3 e aggiunge un nuovo LRC di livello 1 con v_3, e cosi via finché v_{b-1} raggiunge v_b. In questo caso, viene aggiungo un nuovo LRC di livello 2 tra v_1 e v_b. In tal modo, possiamo dire che l'*Hop Level Mechanism* non permette un numero di hop maggiore di b all'interno della rete. Per poterlo implementare, ogni messaggio deve contenere per livello il numero di hop effettuati e il primo nodo che ha inviato il messaggio. Ogni qualvolta viene creato un nuovo livello l, il contatore degli hop dei livelli inferiori di l viene azzerato, viene settato il prossimo nodo come origine del messaggio per tutti i livelli inferiori e viene incrementato il contatore degli hop del livello l.

2. *Hit Count Balancing Mechanism.* Ogni nodo si sceglie un numero prestabilito di LRC. Ogni LRC ha degli *hit counters*, inizializzati a 0 e incrementati ogni qualvolta si utilizza quel LRC. Periodicamente il sistema controlla quali sono gli LRC più utilizzati, che vengono divisi in più LRC, e quelli meno utilizzati, i quali vengono eliminati. Per assicurarci che gli hit counters di tali LRC non divergano, vengono periodicamente divisi per 2, poichè $\sum_{i=0}^{+\infty}(1/2^i) = 2$. Per determinare quali LRC vanno divisi e quali vanno eliminati, si divide lo spazio degli LRC in quattro gruppi a seconda della distanza dei valori degli hit counters dalla loro media stessa:

   - gli LRC con hit counters sopra il doppio della media (*veryhigh group*)
   - gli LRC con hit counters sopra la media, ma inferiori al doppio della media (*high group*)

- gli LRC con hit counters inferiore alla media, ma superiore alla metà della media (*low group*)
- gli LRC con hit counters inferiori alla metà della media (*verylow group*)

Com'è facilmente intuibile, gli LRC che saranno spezzati sono quelli che fanno parte del veryhigh group, mentre simmetricamente quelli del verylow group saranno cancellati.

3. *Small-World Mechanism.* Il meccanismo Small-World si basa sulle *Small World networks*[42], le quali sono grafi particolari in cui i nodi non sono tutti collegati tra loro, ma sono costruite in modo da garantire che ogni nodo sia raggiungibile da un qualsiasi altro in un numero molto basso di passi. L'idea è quindi quella di costruire questi LRC allo stesso modo in cui si costruisce una small world network. Si procede seguendo i seguenti passi:

   - assumendo che lo spazio sia una quadrato, si divide tutto lo spazio in n quadrati.
   - si sceglie un numero massimo q di LRC che si vuole creare, con $q < n$.
   - si fanno q lanci per determinare gli archi LRC che devono essere aggiunti. Per determinare la probabilità che effettivamente un arco tra un nodo U e il centro di un quadrato V ha di essere aggiunto, si usa la formula
     $p_r(U, V) = \frac{d(U,V)^{-r}}{\sum W \neq U d(U,V)^{-r}}$.
     Araùjo e Rodrigues [3] scelgono di settare il valore r a 2.

4. *eCan-like Mechanism.* Questo meccanismo per la creazione di LRC si basa esattamente su quello utilizzato da eCan. L'idea è quella di dividere lo spazio in 4 grandi quadrati (supponendo che lo spazio sia quadrato). Si tracciano quindi due LRC verso due di questi quadrati da ogni nodo. Poi ognuno dei 4 grandi quadrati viene diviso in 4 nuovi quadrati più piccoli e allo stesso modo, si tracciano da ogni nodo 2 LRC verso i quadrati esterni. Questo procedimento è ripetuto per un numero non definito di livelli. A livello effettivo, non avendo quadrati nella rete, quando si dirige un arco da un quadrato a un altro vengono considerati archi tra i nodi responsabili per quelle aree.

3.1.2 Skip Delaunay Network

Restando sempre nel campo delle applicazioni della triangolazione di Delaunay al routing geografico è stato sviluppato Skip Delaunay Network, descritto da Shinji Tsuboi, Tomoteru Oku, Masaaki Ohnishi, Shinichi Ueshima in [38].

Ricordiamo che il motivo per cui le overlay network conosciute non sono adatte alla risoluzione di problemi geografici è che l'identificatore logico associato ai nodi in un overlay network classico è monodimensionale, quindi non si adatta alla rappresentazione delle posizioni geografiche dei nodi. D'altro canto, le triangolazioni di Delaunay soffrono del problema

del largo diametro di rete.
Per risolvere questo problema, Skip Delaunay Network propone la realizzazione di un sistema ibrido tra i due mondi, poiché è composto da:

- una rete logica basata su SkipNet [16], la quale sarà descritta in modo sintetico, nel seguito;
- una triangolazione di Delaunay basata su livelli.

In sintesi, la struttura di Skip Delaunay Network viene costruita seguendo i seguenti punti, per ogni livello l:

- tramite l'utilizzo di una funzione probabilistica, viene deciso se un nodo n deve essere inserito al livello l;
- una volta inseriti tutti i nodi in l, vengono costruite un'overlay network di l utilizzando l'identificatore logico dei nodi e una triangolazione di Delaunay locale di l.

L'ibrido viene quindi realizzato creando due reti parallele: una rete logica, la quale connette per ogni livello un nodo n con un insieme V_n di vicini logici di n, sulla base dell'identificativo logico di ogni nodo, e una rete di Delaunay, la quale viene costruita utilizzando le posizioni dei nodi nella rete e quindi sulla base della vicinanza euclidea dei nodi nello spazio, che quindi connette n con un insieme $DelV_n$ di vicini di Delaunay. Ogni nodo, in questo modo, mantiene un insieme di vicini logici e un insieme di vicini di Delaunay.

Algoritmo

Vanno considerate le seguenti assunzione sui nodi della rete V:

- ogni nodo è uguale agli altri dal punto di vista energetico e del tutto autonomo. Inoltre, assumiamo che questo possieda un identificatore logico e una sua locazione nello spazio.
- ogni nodo possiede una finger table, nella quale memorizza gli identificatori e le posizioni dei nodi adiacenti.

L'algoritmo è iterativo, e continua a generare nuovi livelli finché il nodo n possiede dei vicini nella rete logica. Esso viene diviso in due fasi:

1. fase per la creazione della gerarchia dalla rete logica: In SkipNet, ogni nodo appartiene a un vettore di nodi detto *membership vector*. Tutti i nodi di un membership vector vengono posizionati all'interno di una struttura logica ad anello, la quale servirà per identificare i vicini dei nodi in tale livello. Dato il nodo n appartenente a una struttura

ad anello in un livello i, vengono generate due nuove strutture ad anello al livello $i+1$ (che chiameremo rete livello $(i+1)-0$ e rete livello $(i+1)-1$) dividendo i nodi della struttura del livello i a seconda del primo bit dell'identificatore logico: se iniziano con 0, allora saranno inserite nel primo livello $i+1$, altrimenti nel secondo. Tale procedimento viene iterato finché è possibile dividere tali nodi.

2. fase di creazione della rete di triangolazioni di Delaunay: una volta generata la rete logica a livelli, i nodi appartenenti a ogni struttura ad anello nella rete logica vengono utilizzati per calcolare una rete di triangolazioni di Delaunay. Tale passaggio può essere effettuato in maniera autonoma e distribuita tra gli anelli.

Skip Delaunay Network risolve efficientemente problemi di routing geografico perché si basa su triangolazioni di Delaunay per la consegna dei messaggi. Inoltre, riduce il problema del largo diametro: infatti, quando un nodo v dovrà consegnare un pacchetto a un nodo n, questo analizzerà la gerarchia di triangolazioni di Delaunay create al fine di trovare la via più breve alla destinazione. Questo garantisce un basso numero di hop in presenza di nodi lontani poiché ad ogni livello le celle di Voronoi corrispondenti nello spazio tendono a diventare sempre più grandi (poiché per costruzione sono presenti sempre meno nodi aumentando il livello). Una cella di Voronoi più grande corrisponde ad archi più lunghi nella triangolazione di Delaunay corrispondente.

3.2 Algoritmi localizzati per la costruzione della triangolazione di Delaunay

In questa sezione saranno descritti algoritmi per la costruzione di strutture dati basate su triangolazione di Delaunay applicabili a reti di sensori wireless. Una triangolazione di Delaunay standard non sempre è utilizzabile per le reti di sensori reali perché, in tale contesto, è necessario tener conto del *raggio di trasmissione* dei sensori, il quale può essere inferiore alla distanza tra un nodo n e un suo vicino nella triangolazione. Inoltre, la costruzione delle triangolazioni di Delaunay richiede un alto costo di comunicazione, quindi di difficile applicazione, nella realtà, alle reti di sensori. D'altro canto, sarebbe auspicabile una struttura che, come le triangolazioni di Delaunay, desse supporto ad algoritmi di *routing localizzato*, cioè ad algoritmi di routing in cui un nodo n che deve inviare un messaggio a un nodo d sceglie il nodo a cui inviarlo utilizzando solamente l'informazione locale. Esempi di algoritmi di routing localizzato presentati sono il greedy routing e il compass routing.
Nasce quindi l'esigenza di una nuova struttura calcolabile tenendo conto del raggio di trasmissione dei sensori e in grado di supportare efficientemente routing geografico. Sono state proposte due soluzioni: *Planar Localized Delaunay Triangulation*, proposta in [23] e succes-

sivamente migliorata in [2] al fine di ridurre i costi di comunicazione, e *Restricted Delaunay Graph*[12].

3.2.1 Planar Localized Delaunay Triangulation di Li et al.

Li et al. propongono in [23] un algoritmo per calcolare un grafo planare basato su triangolazione di Delaunay, chiamato *Planar Localized Delaunay Triangulation*, su una rete di sensori V. Tale struttura si presta bene alle reti di sensori, perché è costruita tramite un algoritmo localizzato. Inoltre, la sua struttura garantisce il supporto al greedy routing e il compass routing.

Definizioni matematiche

In generale, per la modellazione di una rete di sensori V è utilizzato lo *Unit Disk Graph* $UDG(V)$.

Definizione 3.2 (Unit Disk Graph)**.** Lo *Unit Disk Graph* $UDG(V)$ di una rete V è un grafo planare in cui esiste un arco tra i nodi solo se la distanza tra loro è al più il raggio di trasmissione. Il raggio di trasmissione dei sensori è detto *unità*.

Si assuma che tutti i sensori della rete abbiano lo stesso raggio di trasmissione. In uno Unit Disk Graph, due nodi sono vicini se e solo se i sensori relativi a questi sono in grado di comunicare tra loro, quindi sono l'uno incluso nel raggio di trasmissione dell'altro. Ad ogni arco viene assegnato un peso, il quale rappresenta un valore normalizzato all'unità della distanza euclidea dei sensori nello spazio: se due sensori hanno distanza inferiore al raggio di trasmissione, esiste un arco tra loro. Li et al. in [23] partono dallo Unit Disk Graph per costruire una struttura che possa essere calcolata tramite algoritmi localizzati in maniera distribuita, col vincolo di limitare il più possibile la crescita dei cammini tra i nodi di $UDG(V)$. Oltre a questo vincolo, la struttura deve verificare le seguenti condizioni:

1. deve essere planare;
2. non può avere archi con peso maggiore di 1.

Poiché la struttura cercata deve avere archi con massimo peso 1, la struttura cercata ha una relazione con lo Unit Disk Graph. In dettaglio, il grafo che stiamo cercando è uno *spanner geometrico* [26] di $UDG(V)$.
Si consideri la seguente definizione:

Definizione 3.3 (Spanner geometrico)**.** Sia $H(V, E_1)$ un grafo su uno spazio euclideo e $V = \{v_1, \ldots, v_n\}$ l'insieme dei nodi del grafo. Sia $G(V, E_2)$ un altro grafo sullo stesso insieme di nodi di H. G è uno *spanner geometrico* di H, o $t-$ *spanner* di H, se presi due nodi

qualsiasi in V, v_i e v_j, la lunghezza del cammino minimo tra v_i e v_j in G è minore o uguale di t volte la distanza euclidea tra v_i e v_j.

La struttura cercata da Li et al. è un t-spanner di $UDG(V)$. Tra i t-spanners di $UDG(V)$, va menzionata la triangolazione di Delaunay privata degli archi con peso maggiore di 1, nota come *Unit Delaunay Triangulation.*

Definizione 3.4. Sia V un insieme di punti in uno spazio euclideo. Si dice *Unit Delaunay Triangulation* $UDel(V)$ di V la triangolazione di Delaunay privata degli archi con peso maggiore di 1. Vale inoltre che:

$$UDel(V) = Del(V) \cap UDG(V)$$

$UDel(V)$ è una struttura che si presta bene alle reti di sensori, perché a differenza delle triangolazioni di Delaunay classiche elimina tutti i nodi non visibili. Il problema è nella sua costruzione: infatti, questa implicherebbe la costruzione di una triangolazione di Delaunay, che è di difficile realizzazione con algoritmi localizzati nonché molto costosa in termini di comunicazione.
Il massimo che si riesce a costruire in maniera localizzata è un sottografo di $UDel(V)$ chiamato *Localized Delaunay Graph* di livello k, $LDel^{(k)}(V)$, dove k rappresenta la distanza in termini di hop dei vicini considerati dal nodo che calcola la triangolazione di Delaunay locale. In dettaglio, nel caso delle reti di sensori con raggio di trasmissione uguale a 1, vale che $k = 1$. $LDel^{(1)}(V)$ è una struttura che è dimostrato da Li et al. in [23] essere un t-spanner di $UDG(V)$. Inoltre, tale grafo possiede archi con lunghezza inferiore a 1, perché considera al momento della costruzione solamente i vicini raggiungibili con un hop, ed è calcolabile in maniera localizzata senza alcun problema. Vale, però, la seguente osservazione:

Osservazione 2. Il Localized Delaunay Graph di livello 1, $LDel^{(1)}(V)$, di un insieme di nodi V in uno spazio bidimensionale non è planare.

Li et al. dimostrano in [23] la non planarità di $LDel^{(1)}(V)$ mostrando un esempio in cui triangoli si intersecano tra loro e con archi. L'algoritmo proposto in [23] prevede una procedura per il calcolo di $LDel^{(1)}(V)$ e una procedura per la ottenere da questo un grafo planare eliminando archi che hanno intersezioni tra loro. Il grafo ottenuto dopo l'esecuzione di questa procedura è il *Planar Localized Delaunay Graph* $PLDel(V)$.

L'algoritmo proposto consiste in due fasi distinte:

1. la prima fase, la quale prevede un algoritmo per la costruzione di $LDel^{(1)}(V)$ in maniera distribuita;

2. la seconda fase, la quale prevede un algoritmo per costruire $PLDel(V)$ a partire da $LDel^{(1)}(V)$.

Algoritmo per la costruzione del Localized Delaunay Graph su V

In questa sezione è descritto l'algoritmo per la costruzione, in maniera distribuita, di $LDel^{(1)}(V)$. Allo stato iniziale dell'algoritmo, ogni nodo $u \in V$ effettua un broadcast della sua identità e della sua posizione, in modo che ogni nodo u venga a conoscenza dei propri vicini $N(u)$ nella rete.
Vengono quindi eseguiti i seguenti passi:

1. il nodo u calcola quindi una triangolazione di Delaunay $Del(N(u) \cup \{u\})$. Si ricordi che $N(u)$ rappresenta l'insieme dei vicini di u raggiungibili con un hop.

2. per ogni arco uv in $Del(N(u) \cup \{u\})$, si considerino i triangoli incidenti in tale arco Δuvw e Δuvz. uv è un arco di Gabriel se entrambi gli angoli in z e in w sono $< \pi/2$. Gli archi di Gabriel sono memorizzati dal nodo u in modo da non essere mai eliminati.

3. il nodo u calcola la lista dei triangoli in $Del(N(u) \cup \{u\})$ aventi tutti gli archi con lunghezza minore di 1. Supponiamo che Δuvz sia uno di questi. Se l'angolo in u è $\geq \pi/3$, u invia un messaggio $proposal(u, v, z)$ per avanzare la proposta di creazione di un nuovo triangolo nella $LDel^{(1)}(V)$ e si mette in ascolto di altri nodi.

4. quando un nodo v riceve un messaggio $proposal(u, v, z)$, v controlla se il triangolo contenuto nella proposta è incluso in $Del(N(v) \cup \{v\})$: se è presente, allora invia in broadcast il messaggio $accept(u, v, z)$, altrimenti il messaggio $reject(u, v, z)$.

5. u aggiunge i lati uv e uw al suo insieme di archi se il triangolo Δuvw è in $Del(N(u))$ e se v e w hanno entrambi inviato messaggi $accept(u, v, w)$ o $proposal(u, v, w)$.

L'algoritmo è abbastanza semplice: ogni nodo costruisce la propria triangolazione di Delaunay sfruttando solamente l'informazione locale. In seguito, ogni nodo salva gli archi di Gabriel di cui fa parte, i quali non saranno mai modificati poiché fanno parte del grafo finale. Infine, viene instaurata una comunicazione tra i nodi per l'accettazione o il rigetto dei triangoli.

Algoritmo per calcolare il Planar Delaunay Graph su V

Alla fine dell'algoritmo descritto nella sezione precedente, la non planarità della struttura costruita cade nei seguenti casi:

- il grafo contiene due triangoli che si intersecano;

- il grafo contiene un triangolo che si interseca con un arco di Gabriel.

In entrambi i casi, la soluzione adottata da Li et al. prevede la ristrutturazione di un triangolo. L'algoritmo per la costruzione di $PLDel(V)$ prevede i seguenti passi:

1. ogni nodo u invia in broadcast tutti gli archi di Gabriel e tutti i triangoli incidenti in u. In questo modo tutti i nodi all'interno del raggio di trasmissione di u vengono informati degli archi di Gabriel e dei triangoli della vista locale di u.

2. si assuma che un nodo $v \in V$ abbia ricevuto informazione sugli archi di Gabriel e su tutti i triangoli dai suoi vicini $N(v)$ tramite i messaggi di broadcast descritti al punto precedente. Presi due triangoli Δuvw e Δxyz, v rimuove il triangolo Δuvw se il suo circumcerchio contiene almeno un nodo tra x, y, z.

3. Ogni nodo u invia in broadcast tutti i triangoli che non sono stati rimossi al passo precedente, e si mette in ascolto di altri nodi.

4. Nella lista degli archi incidenti, un arco uv non viene eliminato se è un arco di Gabriel oppure se è presente il un triangolo Δuvz nella triangolazione locale $LDel^1(u)$ tale che non sia stato eliminato dalle triangolazioni locali di v e z.

Il grafo ottenuto è planare, t-spanner di $UDG(V)$ e supporta routing geografico. Questo è proprio $PLDel(V)$. Vale il seguente teorema:

Teorema 3.2.1.
$PLDel(V)$ è un grafo planare ed è un $\frac{4\sqrt{3}}{9}\pi$spanner dello Unit Disk Graph su V.

3.2.2 Fast Localized Delaunay Triangulation

Come Li et al. in [23], F. Araùjo et al. in [2] si pongono il problema di trovare una struttura calcolabile tramite l'utilizzo di algoritmi localizzati [35] con supporto ad algoritmi di routing localizzato. Basandosi sul lavoro di Li et al., Araùjo et al. descrivono un algoritmo in grado di costruire $PLDel(V)$ con un costo di comunicazione inferiore. Il fatto che il costo di comunicazione sia inferiore fa si che tale algoritmo sia maggiormente utilizzabile nella realtà.
L'algoritmo proposto è chiamato *Fast Localized Delaunay Triangulation*. Le modifiche apportate da Araùjo et al. all'algoritmo descritto in [23] riguardano principalmente i messaggi utilizzati in fase di creazione di $LDel^{(1)}(V)$ e il raggruppamento dei messaggi inviati.

I punti di forza di Fast Localized Delaunay Triangulation sono i seguenti:

1. se definiamo come passo di comunicazione il periodo richiesto per l'invio e la recezione di uno o più messaggi che non sono casualmente correlati tra loro, costruisce $PLDel(V)$ con un solo passo di comunicazione, a differenza dell'algoritmo di Li et al. che ne utilizza 4;

2. è applicabile alle reti dinamiche;

3. è localizzato, quindi adatto alle reti di sensori wireless;

4. ogni nodo tiene traccia solamente di un insieme costante di vicini (proprietà derivata dalla triangolazione di Delaunay);

5. il grafo costruito ha una buona densità.

Algoritmo

L'algoritmo Fast Local Delaunay Triangulation è totalmente decentralizzato e localizzato. Esso viene diviso in tre fasi logiche:

1. la fase di ricerca dei vicini;

2. la fase di triangolazione;

3. la fase di correzione;

4. la fase degli archi di Gabriel;

La fase di ricerca dei vicini, come suggerisce il nome, rappresenta la fase in cui ogni nodo $u \in V$ viene a conoscenza dei propri vicini. Se consideriamo una rete di sensori statica, come nell'algoritmo di Li et al. i nodi vengono a conoscenza dei propri vicini utilizzando dei messaggi di broadcast iniziali. Per la gestione della dinamicità della rete vengono invece utilizzati messaggi *BEACON* e messaggi *TRIANGULATE* periodicamente scambiati tra i nodi della rete per venire a conoscenza di eventuali cambiamenti.
La fase di triangolazione consiste nella costruzione della triangolazione di Delaunay locale dei nodi. Durante tale fase, ogni nodo $u \in V$ costruisce $Del(N(u) \cup \{u\})$ e comunica con i propri vicini utilizzando messaggi *TRIANGULATE*.
Per semplicità di notazione, si consideri il predicato $Delaunay\ \Delta_p(q, r)$, il quale vale $true$ se il triangolo pqr è incluso in $Del(N(p) \cup \{p\})$. Se $Delaunay\ \Delta_p(q, r) = true$ e l'angolo $qpr < \pi/3$, p invia messaggi *TRIANGULATE* per notificare i vicini della creazione del triangolo. Il controllo sull'angolo viene effettuato solamente con lo scopo di limitare il numero di messaggi *TRIANGULATE* nella rete. Alla fine della fase di correzione, la struttura costruita corrisponde a un supergrafo di $LDel^{(1)}(V)$.

La terza fase è la cosiddetta fase di correzione, la quale ha lo scopo di eliminare archi non consistenti nella triangolazione di Delaunay. Alla fine della seconda fase, ogni nodo u nella rete avrà calcolato la propria triangolazione locale $Del(N(u) \cup \{u\})$ e avrà ricevuto un insieme di messaggi *TRIANGULATE* dai propri vicini. Durante la fase di correzione, i nodi utilizzano le informazioni nei messaggi *TRIANGULATE* ricevuti al fine di eliminare inconsistenze dalla rete. La triangolazione calcolata da ogni nodo u durante la fase 2 dell'algoritmo può essere invalidata per due motivi:

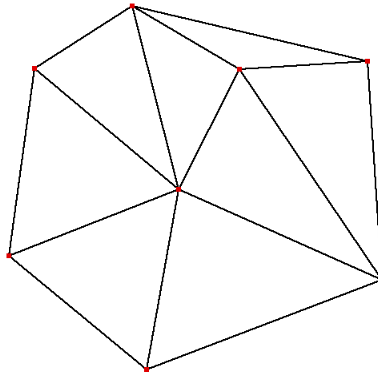

Figura 3.3: La somma degli angoli in un nodo interno è $2 \times \pi$, quindi limitando l'invio di messaggi *TRIANGULATE* al caso dell'angolo inferiore a $\pi/3$, si limita il numero di tali messaggi per nodo a 6.

.

1. esiste un nodo w nella rete tale che il circumcerchio del triangolo pqr contiene w. In questo caso il nodo q o il nodo r hanno inviato un messaggio triangulate al nodo w che invalida il triangolo pqr. Questo caso sta alla base delle triangolazione di Delaunay, ed elimina tutti gli archi che non appartengono a questa.

2. un qualche nodo w invia un messaggio per un triangolo wuz che interseca pqr e u o z invalidano il triangolo pqr (poiché uno dei due sta nel circumcerchio di pqr). Questo caso previene le intersezioni.

Fase degli archi di Gabriel

Questa fase ha lo scopo di aggiungere gli archi di Gabriel mancanti al grafo. Non vengono però aggiunti tutti i Gabriel edges possibili: infatti, un Gabriel edge pq non viene aggiunto da p se nessun predicato *Delaunay* $\Delta_p(q, r)$ vale.

L'algoritmo calcola lo stesso grafo di PLDT

F. Araùjo et al. dimostrano in [2] che il grafo calcolato dall'algoritmo Fast Localized Delaunay Triangulation è $PLDel(V)$. La dimostrazione si basa sul fatto che, dopo la seconda fase dell'algoritmo, essenzialmente è stato costruito globalmente un supergrafo di $LDel^{(1)}(V)$, poiché ogni nodo ha calcolato una triangolazione locale. Tale grafo non è planare, poiché le inconsistenze verranno eliminate solamente durante la fase 3. La fase di correzione consiste nell'eliminazione degli archi:

- che non appartengono a $LDel^{(1)}(V)$, poiché non soddisfano la triangolazione di Delaunay;
- che appartengono ai triangoli che si intersecano in $LDel^(2)(V)$.

Quindi, viene costruito un sottografo di $LDel^{(1)}(V)$ che è planare ($LDel^{(1)}(V)$ non è planare). Tale grafo è proprio $PLDT(V)$.

3.2.3 Restricted Delaunay Graph

L'ultimo algoritmo localizzato applicato alle reti di sensori wireless che consideriamo consiste nella costruzione di *Restricted Delaunay Graph* (*RDG*), ed è descritto da J. Gao et al. in [12]. Come gli algoritmi di Li et al. [23] e Araùjo et al [2], l'algoritmo di Gao et al. ha lo scopo di costruire in maniera localizzata una struttura in grado di supportare efficientemente problemi geografici applicati alle reti di sensori wireless ed è caratterizzato dalla definizione di una clusterizzazione dei nodi.
RDG, come $PLDT(V)$, limita il numero di hop effettuati da algoritmi di routing, poiché è anche esso uno spanner geometrico di $UDG(V)$. Di seguito viene presentato l'algoritmo, il quale è diviso in due fasi distinte:

1. la prima fase, detta *fase di clustering*, consiste nel raggruppare i nodi in cluster. *RDG* sarà costruito solamente tra i cluster, mentre i nodi all'interno di questi fanno riferimento a dei nodi speciali per l'invio dei messaggi attraverso i cluster;
2. la seconda fase, la quale consiste nella costruzione effettiva di *RDG* sui nodi speciali dei cluster.

Il punto di forza dell'algoritmo rispetto a quelli trattati fino ad ora è che è in grado di mantenere in maniera distribuita la correttezza in caso di movimento di nodi. La debolezza è che utilizza un sistema centralizzato per il suo funzionamento.

Fase di Clustering

In questa prima fase vengono creati dei cluster in tutto l'insieme iniziale V dei nodi della rete. I vari nodi della rete, alla fine del procedimento, saranno divisi in tre gruppi che definiscono tipi di nodi non esclusivi tra loro:

- i *clusterhead nodes*, nodi a capo di un cluster, i quali identificano il cluster e vengono utilizzati per il calcolo di *RDG*
- i *gateway nodes*, nodi che hanno conoscenza di altri nodi in altri cluster (poiché sono vicini in $UDG(V)$) e vengono utilizzati dai cluesterhead nodes per inviare messaggi attraverso i cluster.

- i *client nodes*, nodi all'interno della rete che non partecipano alle operazioni tra i cluster. Sono tutti i nodi che non sono ne clusterhead ne gateways nodes.

Mentre i gateway nodes dipendono dalla costruzione dei cluster e quindi non possono essere scelti, i clusterhead nodes vengono scelti utilizzando un algoritmo a livelli. A ogni livello:

1. ogni nodo incluso nella lista dei partecipanti alla selezione del clusterhead sceglie un clusterhead all'interno del proprio raggio di visione. Al livello i, il raggio di visione di un nodo v è $\frac{2^i}{\log n}$.
2. la lista di partecipanti per il livello $i + 1$ è aggiornata con i nodi che sono stati scelti al livello i.

Alla fine del procedimento, ci saranno una serie di nodi scelti, i clusterhead, e una serie di nodi che ha scelto il clusterhead c direttamente o indirettamente, i quali rappresentano i client che insieme a c comporranno il cluster di c.
I nodi in cluster diversi che riescono a interagire sono invece i nodi gateway, i quali saranno utilizzati per le comunicazioni tra clusters. Ogni coppia di cluster avrà solamente una coppia di nodi gateway, nonostante sia possibile avere più nodi che soddisfano il requisito. Questi altri nodi, se presenti, sono *candidati a nodi gateway.*

Calcolo di RDG

La seconda fase dell'algoritmo riguarda la costruzione della Restricted Delaunay Graph, che risulta ristretta in quanto definita solamente tra i nodi che definiscono i cluster nella nostra rete. Consideriamo quindi il sottografo G_{CG} del grafo iniziale con solamente tali nodi. J. Gao et al. [12] costruiscono un RDG invece che utilizzare la triangolazione di Delaunay per due motivi che rendono tale triangolazione una soluzione insoddisfacente per il problema delle reti ad hoc:

1. la triangolazione di Delaunay non riesce a rappresentare il limite del raggio di visione di un dispositivo ad hoc mobile. Per fare ciò, bisogna limitare la triangolazione eliminando tutti gli archi che hanno dimensione maggiore di un' *unità.*
2. la condizione principale che fa in modo che una triangolazione sia "di Delaunay" è che la condizione del cerchio vuoto valga su tutti i nodi della rete. Questo non è facilmente verificabile su una rete in cui sia ha solamente una visione locale.

Per questi motivi, viene definito RDG:

Definizione 3.5 (Restricted Delaunay Graph)**.** Si definisce col nome di *Restricted Delaunay Graph*(RDG) di un insieme di punti V, un grafo planare che contiene tutti gli archi di Delaunay di lunghezza inferiore o uguale a un'unità.

Gli archi di Delaunay in una RGD sono chiamati *short Delaunay edges*. Una RDG viene quinndi costruita , considerando il sotto-grafo del grafo generato a partire dalla triangolazione di Delaunay utilizzando solo i nodi clusterhead e gateway e solamente short Delaunay edges. Viene dimostrato che tale grafo è un t-spanner dello spazio euclideo (secondo la distanza euclidea) e, in particolare, dello spazio topologico, cioè per coppia di nodi u, v all'interno del grafo, vale $d_{G_{CG}}(u, v) \leq C \times d(u, v)$ dove $d(.)$ rapprensenta il numero di hop ottimo nello spazio topologico.

Mantenimento della rete

J. Gao et al. [12] descrivono un algoritmo per il mantenimento di *RDG* durante il movimento dei nodi nella rete. Questo rappresenta un punto di forza rispetto ad algoritmi che non gestiscono questa situazione, poiché è molto probabile che questo scenario si verifichi nella realtà, se consideriamo dispositivi mobile ad hoc.
Per mantenere la struttura *RDG*, ogni nodo u (clusterhead e gateway) tiene in memoria la lista degli archi incidenti, chiamata $E(u)$. Affinché il grafo sia calcolato correttamente, è necessario che siano verificati i seguenti quattro punti:

1. ogni arco in $E(u)$ deve essere uno short Delaunay edge;
2. se un arco presente in $E(u)$ collega u con un suo vicino v, allora questo deve essere presente anche in $E(v)$;
3. il grafo complessivo deve essere planare;
4. l'unione degli $E(u)$, $\forall u$, deve contenere tutti i short Delaunay edges.

Per garantire questi quattro punti, per prima cosa il nodo u viene a conoscenza dei suoi vicini $N(u)$ tra i clusterhead e gateways, e calcola la sua triangolazione di Delaunay locale. Questa triangolazione, comunque, essendo locale potrebbe contenere dei lati che *globalmente* non sono di Delaunay, i quali andranno a rendere il nostro grafo globale non planare. Inoltre potrebbero esserci delle inconsistenze. Per risolvere tale situazione, il nodo u invia messaggi contenenti la propria triangolazione locale ai nodi clusterhead e gateway vicini. Ognuno di questi nodi, quindi, risolve le inconsistenze eliminando dall'insieme di archi di u gli archi inconsistenti con la propria visione locale. Questa procedura garantisce anche che non saranno creati archi che si incrociano nel grafo finale. Questo viene garantito dal seguente lemma:

Lemma 3.2.2. *Per ogni coppia di nodi visibile vw e uz, se gli archi vw e uz si incrociano, allora uno dei 4 nodi vede tutti gli altri 3.*

Il fatto che uno dei nodi veda gli altri nodi garantisce che questo sia in grado di calcolare correttamente una triangolazione di Delaunay, quindi che non ci siano archi che si incrociano tra loro (poiché la triangolazione è planare).

	PLDel(V)[23]	FLDT [2]	RDG [12]
Supporto churn	No	Si	Si
Supporto nodi in movimento	No	No	Si
Supporto greedy routing	Si	Si	Si
Supporto compass routing	Si	Si	Si
Supporto routing geografico	Si	Si	Si
Completamente distribuito	Si	Si	No

Tabella 3.1: Algoritmi localizzati a confronto

Mantenimento dei nodi gateway

Dal momento in cui sono permessi movimenti dei nodi, è possibile che un nodo gateway non possegga più, dopo un movimento, le caratteristiche per essere considerato tale. Viene quindi presentato un algoritmo che permette ai nodi clusterhead di scegliere i nodi gateway in maniera appropriata. Questo algoritmo deve essere eseguito solamente quando un nodo non può più vedere un altro, quindi quando effettivamente c'è un allontanamento tale da far cambiare l'overlay globale.

Per ogni coppia di clusterhead c_1, c_2, definiamo un *bipartite graph* avente per nodi l'insieme dei nodi dei cluster di c_1 e c_2, e per archi tutti gli archi pq tale che p è un nodo che sta nel cluster di c_1 e q è un nodo che sta nel cluster di c_2. Avevamo definito i nodi del grafo bipartito col termine di *candidato a gateway node*.

3.2.4 Conclusioni

Nelle precedenti sezioni sono stati analizzati tre algoritmi per la costruzione in maniera localizzato di topologie in grado di supportare routing geografico e applicabili alle reti di sensori wireless.

La Tabella 3.1 mostra le principali differenze tra gli algoritmi localizzati e le topologie calcolate. La caratteristica di tutte le topologie è che mantengono il supporto al greedy e al compass routing, ed essendo basate tutte su una struttura in spazio geometrico non perdono il supporto al routing geografico.
La costruzione di $PLDel(V)$ è di natura distribuita, quindi non richiede il supporto di unità centralizzate, al contrario di RDG che ne fa uso per il mantenimento dei cluster e delle categorie di nodi (clusterhead, gateway).
Al contrario, RDG da supporto alle reti dinamiche, in cui i nodi possono cambiare la loro posizione. Questa caratteristica è di grande importanza poiché, come specificato più volte, la costruzione delle triangolazioni di Delaunay è costosa in termini di messaggi. Il supporto

a reti dinamiche permette a RDG di non dover essere ricostruito a ogni cambiamento nella rete.

3.3 Algoritmi di supporto agli Ambienti Virtuali Distribuiti

In questa sezione sono presentati due algoritmi distribuiti per la costruzione di una triangolazione di Delaunay applicabile ad ambienti virtuali distribuiti. Gli ambienti virtuali distribuiti rappresentano una sfida perché devono garantire che la visione da parte di ogni utente connesso a tale ambiente sia consistente con quella degli altri utenti, e che le azioni compiute all'interno dell'ambiente virtuale siano percepite come azioni in tempo reale. Per questo motivo, è necessario che il numero di messaggi all'interno della rete sia limitato in qualche modo. Inoltre, è necessario un supporto per l'alta dinamicità di tali reti: utenti si connettono, disconnettono e cambiano posizione molto frequentemente negli ambienti virtuali.
In dettaglio, una prima limitazione del numero di messaggi della rete è dettata dal fatto che i nodi hanno interesse solamente nei cambiamenti locali: di conseguenza, non è necessario inviare a un nodo l'informazione di qualcosa che è cambiata ai punti opposti della rete. L'area entro il quale il nodo n deve ricevere aggiornamenti e cambiamenti relativi alla rete è detta *Area di Interesse* di n.

Gli algoritmi presentati sono l'algoritmo di Buyukkaya et al. [8], che riduce il numero di messaggi nella rete rilassando i vincoli della proprietà equiangolare della triangolazione di Delaunay, e l'algoritmo di Ghaffari et al. [14], il quale realizza una struttura composta da triangolazioni di Delaunay parallele al fine di gestire l'alta dinamicità degli ambienti virtuali.

3.3.1 L'algoritmo di Buyukkaya et al.

In questa sezione, viene descritto l'algoritmo di E. Buyukkaya e M. Abdallah [8], il quale definisce una struttura distribuita basata su triangolazione di Delaunay per il supporto agli ambienti virtuali distribuiti. Buyukkaya et al. propongono la costruzione di una nuova struttura ottenuta *rilassando* il vincolo della proprietà equiangolare della triangolazione di Delaunay. Quindi, in determinate aree limitano il flip degli archi distinguendo le azioni da compiere a seconda della fase che causerebbe il flip: fase di join di un nodo, movimento di un nodo o leave di un nodo.
Limitando i flip degli archi definiscono una limitazione sul numero di messaggi che viaggia nella rete.

Join di un nodo

Durante la fase di Join di un nodo n si può verificare un numero non definito di operazioni di flip degli archi della triangolazione, che dipendono dalle posizioni dei nodi e dalla triango-

lazione stessa. Al fine di imporre una limitazione su tale numero, viene definita l'*Angle Area* di un triangolo T $AA(T)$ come una proiezione P_T del triangolo T nella rete tale

$$P_T = kT$$

con $k > 1$. La angle area di T viene utilizzata per determinare il nodo a cui inoltrare la richiesta di Join di un nodo n. In particolare, la fase di Join di n si compone di due step:

- la ricerca del triangolo che contiene il nodo n;
- l'esecuzione delle operazioni di flip degli archi, le quali devono essere in numero limitato rispetto a quelle triangolazione di Delaunay originale.

La rete mantiene in memoria una lista di nodi speciali a cui i nodi che effettuano la fase di join fanno riferimento per inserirsi nella rete, chiamati *nodi gate*. All'ingresso nella rete, il nodo n contatta un nodo gate p, il quale inizia la fase di ricerca del triangolo che contiene n nella rete. Inizialmente, p controlla tra i triangoli della sua triangolazione se ne esiste uno che include n: se esiste, allora si passa alla fase successiva della Join, altrimenti p usa le Angle Areas dei suoi triangoli T_i per trovare quella che include n, detta $AA(T_j)$. Tale area esiste perché è possibile scegliere arbitrariamente il valore k da utilizzare nella proiezione. Infine, p sceglie tra i suoi vicini $v \in T_j$ il nodo a cui inoltrare la richiesta di Join di n, scegliendo il nodo con distanza euclidea inferiore da n. L'algoritmo procede ricorsivamente finché non viene trovato il triangolo T che contiene n.
La seconda fase relativa alla Join riguarda le operazioni di flip degli archi: sia T_1 il tringolo che contiene n nella triangolazione costruita al tempo t. In questa fase T_1 viene diviso in tre sottotriangoli, creando 3 nuovi archi da n ai vertici di T_1. La limitazione alle operazioni di flip degli archi è data dal fatto che la verifica della proprietà equiangolare della triangolazione di Delaunay è limitata geograficamente ai 3 triangoli che possiedono un lato in comune con T_1.

Cambio di posizione di un nodo

Le operazioni di flip degli archi si possono verificare anche nel caso di un movimento di un nodo. È facile immaginare una situazione del genere: la triangolazione di Delaunay massimizza l'angolo minimo dei triangoli, ma un movimento di un qualunque nodo libero di muoversi cambia tale angolo.
Buyukkaya et al. descrivono una procedura per limitare le operazioni di flip anche se generate da operazioni di movimento dei nodi della rete virtuale.
In dettaglio, viene definita la *Flip-Free Area* di un nodo n.

Definizione 3.6 (Flip-free area)**.** Si dice *Flip-Free Area* di un nodo n in una triangolazione $DT(V)$ di una rete V l'area formata dall'insieme dei triangoli $\{T_i\} \in DT(V)$ tali che n è un vertice di T_i.

In una Flip-Free Area, ogni nodo n della rete è libero di effettuare movimenti senza generare operazioni di flip degli archi. Le operazioni di flip degli archi della triangolazione dovuta al movimento dei nodi si verificano quindi soltanto quando i nodi oltrepassano la *baseline* di uno dei triangoli di cui sono vertici.

Leave di un nodo

Infine, Buyukkaya et al. forniscono un algoritmo per i nodi che vogliono eseguire la disconnessione volontaria dalla rete. In dettaglio, quando un nodo n si sta per disconnettere, effettua un ciclo sui triangoli $\{T_i\}$ di cui è vertice effettuando operazioni di flip dei lati di cui n fa parte, finché possibile. Quando non sono possibili flip di lati, lo stato della vista locale di n comprende:

- 3 vicini: in questo caso, se a, b e c sono i 3 vicini di n, viene creato il nuovo triangolo Δabc, n elimina i suoi triangoli rimanenti ed esce dalla rete.
- 4 vicini, nel caso in cui siano presenti nodi cocircolari a 4 a 4 nella rete: in questo caso, se a, b e c e d sono i 4 vicini di n, vengono creati due triangoli con i 4 nodi (non ha importanza quale sia il lato diagonale del poligono composto da a, b e c e d, poiché in entrambi i casi possibili la triangolazione è corretta), n elimina i suoi triangoli rimanenti ed esce dalla rete.

In una rete bidimensionale, i casi elencati sono gli unici possibili.

3.3.2 L'algoritmo di Ghaffari et al.

Mohsen Ghaffari, Behnoosh Hariri e Shervn Shirmohammadi descrivono in [14] un'architettura distribuita per il supporto di Massive Multiplayer Virtual Environments (MMVE). I MMVE sono ambienti virtuali dove milioni di utenti connessi in tutto il mondo possono interagire tra loro. Esempi di MMVE possono essere i giochi di ruolo online (MMPORG) e ambienti virtuali social.

Come descritto nella sezione precedente, la principale sfida è la costruzione di una struttura che sia in grado di gestire una rete altamente dinamica, poiché gli utenti effettuano molto frequentemente connessioni, disconnessioni e cambiamenti di posizione.

A differenza dell'algoritmo di Buyukkaya et al., Ghaffari et al. in [14] propongono l'utilizzo di due triangolazioni di Delaunay parallele per la gestione degli aggiornamenti e dell'alta dinamicità della rete. Queste due triangolazioni parallele utilizzano il greedy routing per potersi aggiornare correttamente a tempi alterni: per esempio, se la prima si aggiorna ai tempi 0T, 2T, 4T, allora la seconda si aggiorna ai tempi T, 3T, 5T. L'utilizzo di due triangolazioni

di Delaunay parallele è necessario per garantire che l'algoritmo di greedy routing sia libero da cicli. Utilizzando una sola triangolazione di Delaunay, infatti, non sarebbe possibile l'utilizzo del greedy routing per permettere ai nodi di comunicare eventuali cambiamenti di posizione, poiché la modifica apportata alla triangolazione nel momento in cui il cambiamento di posizione è avvenuto non garantisce più che questa sia di Delaunay, quindi il supporto al greedy routing. Ghaffari et al. propongono di dividere i nodi della rete V in due insiemi R(Red) e B(Black) e di mantenere due triangolazioni di Delaunay parallele: $DT(R)$ e $DT(B)$. I due insiemi aggiornano le proprie informazioni a tempi che differiscono di $t/2$ utilizzando l'altro grafo per effettuare un greedy routing: per esempio, se al tempo t i nodi in R aggiornano la propria posizione inviando messaggi a nodi in B, al tempo $t + t/2$ saranno i nodi in B ad aggiornarsi utilizzando il grafo costruito su R.
In questo modo, non essendo il grafo costruito nell'insieme B ancora stato aggiornato, è garantito che la triangolazione di Delaunay $DT(B)$ sia valida al momento in cui sono inviati i messaggi. I nodi che fanno parte degli insiemi B e R possono essere scelti arbitrariamente, anche se è consigliato l'utilizzo di una distribuzione uniforme, il quale può influire positivamente sulla velocità di convergenza dell'algoritmo.

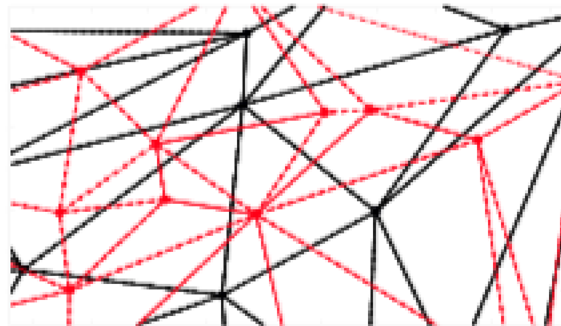

Figura 3.4: Struttura Red-Black

.

Per semplicità, in questa sessione è descritto solo l'aggiornamento di posizione del grafo dei nodi nell'insieme R, poiché quello dei nodi nell'insieme B utilizza la stessa procedura.

Si consideri la seguente definizione di *Vertex Region* di un nodo n:

Definizione 3.7. Si consideri un grafo $G = (V, E)$ nell'insieme di punti V, dove per ogni $v_i \in V$, $N(v_i)$ rappresenta l'insieme di vicini di v_i. Per ogni $v_i \in V$, definiamo la *Vertex Region* di v_i nel grafo G, $VR_G(v_i)$, come l'insieme di tutti i punti del piano che sono più vicini a v_i rispetto a qualunque altro punto in $N(v_i)$

L'aggiornamento della posizione dei nodi in R corrisponde al loro aggiornamento della Vertex Region. L'algoritmo di aggiornamento dei nodi in R può essere diviso in due fasi:

1. i nodi in R comunicano la loro posizione ai vecchi vicini in R: in questo modo, ogni nodo $r \in R$ sarà in possesso dell'informazione necessaria all'aggiornamento di $VR_R(r)$ con le nuove posizioni dei nodi in R;

2. tra le nuove Vertex Regions dei nodi in R calcolate, sono identificate quelle che si sovrappongono. I nodi relativi a tali regioni diventeranno nuovi vicini nella nuova rete calcolata.

Per trovare le Vertex Regions sovrapposte, le informazioni di ogni Vertex Region in R $VR_R(r)$ è inviata ai nodi b in B tali che $VR_B(b)$ contiene almeno un punto in comune con $VR_R(r)$. Supponendo che il nodo $v_i^b \in B$ trovi almeno un punto x in comune tra due VR rosse, dove $x \in VR_B(v_i^b)$, v_i^b sarà responsabile del comunicare al primo nodo rosso di aggiungere un collegamento con il secondo nodo.

I nodi in B responsabili della Vertex Region di un nodo rosso v_j^r sono detti Black Parents di v_j^r.

Definizione 3.8 (Black Parents)**.** Dato un nodo $v_j^r \in R$, definiamo *Black Parents* di v_j^r l'insieme $BP(v_j^r) = \{v_i^b | VR_B(v_i^b) \cap VR_R(v_j^r) \neq \emptyset\}$

L'invio della Vertex Region $VR_R(v_j^r)$ di v_j^r ai nodi dell'insieme $BP(v_j^r)$ viene diviso in due fasi:

1. la prima fase consiste nell'inviare il messaggio contenente la Vertex Region di v_j^r a un nodo in particolare in $BP(v_j^r)$. In dettaglio, se $v_j^r \in VR_B(v_j^b)$ allora $v_j^b \in BP(v_j^r)$. Per questo motivo, ogni nodo in R memorizza il nodo vicino più vicino appartenente all'insieme B e utilizza questo per l'invio del messaggio contenente la Vertex Region calcolata;

2. la seconda fase consiste nel broadcasting del messaggio contenente la Vertex Region di v_j^r tra i Black parents di v_j^r. Questa fase consiste nell'invio del messaggio da parte di un nodo n che ha ricevuto il messaggio ad ogni nodo d in B tale che la Vertex Region $VR_B(d)$ abbia dei punti in comune con la Vertex Region contenuta nel messaggio.

Infine, viene calcolata la struttura utilizzando le informazioni acquisite nelle fasi precedenti: una volta che le Vertex Regions di due nodi in R r_1 e r_2 sono state calcolate dai Black Parents di uno dei due nodi, per esempio di r_1, uno dei nodi in $BP(r_1)$ invia a r_1 un messaggio di candidate neighbor per r_2.
In questo modo, i nodi r in R vengono a conoscenza dei loro candidate neighbors. L'algoritmo per il calcolo dei nuovi vicini effettivi è il seguente:

1. il nodo r_1 considera la bisettrice ortogonale del segmento che lo collega a un suo candidate set r_2;

2. se la bisettrice ortogonale interseca $VR(r_1)$, allora significa che esiste una porzione di $VR(r_1)$ che è più vicina a r_2 rispetto che a r_1: in questo caso, r_1 aggiorna $VR(r_1)$ considerando solamente il suo sottoinsieme più vicino a r_1, e aggiunge r_2 ai suoi vicini.

3. l'aggiornamento di $VR(r_1)$ può generare la creazione di vicini ridondanti per r_1. Questi vicini possono essere identificati utilizzando la bisettrice ortogonale del lato che li collega a r_1: se tale bisettrice è fuori da $VR(r_1)$, allora il vicino è ridondante e l'arco che lo collega a r_1 viene eliminato

Una volta calcolata la lista dei nuovi vicini, il nodo r_1 si mette in contatto con questi utilizzando un messaggio *HelloNeighbor*.

L'aggiornamento della triangolazione globale sull'insieme $V = R \cup B$ può essere effettuato in questo modo: un nodo $r \in R$ invia la propria Vertex Region aggiornata ad ogni nodo $b \in BP(r)$. Ognuno di questi controlla se la bisettrice ortogonale del segmento che lo collega a r interseca internamente la Vertex Region di r, e in tal caso invia a r una richiesta di vicinato contenente le informazioni sulla sua posizione. Inoltre, ognuno dei nodi b ricostruisce la sua triangolazione globale considerando r come vicino.

3.3.3 Conclusioni

Nelle ultime due sezioni sono stati analizzati due algoritmi distribuiti con applicazione agli ambienti virtuali distribuiti: l'algoritmo di Buyukkaya et al. e l'algoritmo di Ghaffari et al.
Il primo descrive una serie di procedure per gestire efficientemente gli ambienti virtuali distribuiti limitando il numero di messaggi che vengono inviati nella rete senza perdere le performance richieste, poiché considera la triangolazione di Delaunay troppo vincolante per un ambiente virtuale distribuito, in cui ad ogni nodo importa conoscere l'aggiornamento solamente della porzione di mondo intorno a lui.
Il secondo costruisce una triangolazione di Delaunay distribuita su una rete ad alta dinamicità, sfruttando delle triangolazioni di supporto per l'invio di messaggi utilizzando algoritmi di routing efficienti.

La Tabella **??**ostra le principali differenze trovate durante l'analisi: l'algoritmo di Buyukkaya si presta meglio agli ambienti virtuali perché si pone il problema della limitazione del numero di messaggi nella rete, osservando correttamente che in un ambiente virtuale distribuito non è di grande rilevanza per un nodo di aggiornare la triangolazione correttamente poiché i messaggi di aggiornamento nella rete interessano nodi vicini. La mancanza del supporto ad algoritmi di routing di tipo greedy o compass dell'algoritmo di Buyukkaya et al.

	Buyukkaya et el. [8]	Ghaffari et al. [14]
Supporto churn	Si	Si
Supporto alta dinamicità di rete	Si	Si
Limitazione messaggi	Si	No
Supporto greedy routing	No	Si
Completamente distribuito	No (nodi gate)	Non specificato

Tabella 3.2: Differenze tra algoritmi per Ambienti Virtuali Distribuiti

risulta essere giustificata in questo contesto, perché tale supporto esiste entro l'area di interesse dei nodi, e questo è sufficiente per l'applicazione dell'algoritmo. Entrambi gli algoritmi analizzati hanno supporto all'alta dinamicità della rete e al churn: questo è ragionevole per via della natura degli ambienti virtuali distribuiti.

3.4 Algoritmi orientati a reti distribuite e P2P

In questa sezione sono presentati alcuni algoritmi distribuiti per la costruzione della triangolazione di Delaunay con applicazioni alle reti distribuite e P2P. In particolare, è presentato l'algoritmo di Liebeherr et al. [24], un algoritmo incrementale per applicazioni di tipo multicast, l'algoritmo di Steiner et al. [33], che si differenzia dagli altri perché comprensivo di procedure per la costruzione di una triangolazione di Delaunay in uno spazio a d-dimensioni, e l'algoritmo di Ohnishi et al.[27], con applicazione ai Virtual Collaborative Spaces.

3.4.1 L'algoritmo distribuito di J. Liebeherr et al.

Liebeherr descrive in [24] un algoritmo incrementale per costruire una triangolazione di Delaunay in maniera distribuita in modo da poter utilizzarla come overlay network per applicazioni di tipo *multicast*. La scelta di Liebeherr in un'overlay basato su triangolazioni di Delaunay è motivato dal fatto che la triangolazione gode delle seguenti proprietà:

1. la triangolazione di Delaunay ha una serie di vie non sovrapposte per ogni coppia di nodi della rete;

2. il numero di archi da un nodo in una triangolazione di Delaunay è in media 6;

3. una volta che la rete è costruita, l'invio di un pacchetto è gestito dall'overlay senza il bisogno di utilizzare un protocollo di routing.

Si consideri una rete V: ad ogni nodo n in V è assegnato un *identificatore logico* e un *identificatore fisico*. L'identificatore logico è un codice univoco che viene realizzato in fase di configurazione della rete, e che somiglia a una serie di coordinate logiche spaziali. L'identificatore fisico è qualcosa di prestabilito che viene creato tramite una fusione tra l'indirizzo IP

della macchina e il numero di porta UDP.
Viene definito un ordinamento tra gli indirizzi logici, sulla base dell'algoritmo Divide et Impera di Guibas e Stolfi (Sezione 2.2). Si consideri la seguente proposizione:

Proposizione 3.4.1 (Ordinamento nella rete)**.** *Si considerino due nodi a e b appartenenti a una rete di nodi V: si dice che $a < b$ se $y_a < y_b$ oppure $y_a = y_b$ e $x_a < x_b$, dove $a = (x_a, y_a)$ e $b = (x_b, y_b)$.*

L'algoritmo utilizza un sistema di *rendez-vous* con un server per permettere ai nodi che non appartengono alla rete di conoscere almeno un nodo che già ne fa parte. Questo meccanismo viene utilizzato in fase di join di un nuovo nodo n permettere la ricerca nella rete del nodo v più vicino a n tramite greedy routing.

L'algoritmo è *temporizzato*, in quanto utilizza dei timer per stabilire in quale momento la rete deve essere aggiornata inviando dei messaggi tra i nodi.

Alcuni tra i timer utilizzati sono:

- l'*Hearthbeat timer*, il quale stabilisce quando i nodi devono inviare i messaggi ai propri vicini. Può essere *fast* nel caso in cui un nodo goda dello stato di nodo *Leader* (il significato sarà spiegato nella prossima sezione), oppure *slow*. Anche il server possiede un Hearthbeat timer, il quale viene utilizzato per aggiornare la propria cache

- il *Neighbor timer*, allo scadere del quale ogni nodo controlla se ha ricevuto messaggi dal vicino x e, eventualmente, elimina x dalla tabella dei vicini, dando per scontato che x non partecipi più in modo attivo alla rete.

- il *Cache timer*, allo scadere del quale il server, il quale ha inviato un messaggio di *CachePing* a un nodo N e non ha ricevuto risposta da n, elimina n dalla propria cache.

- il *Leader timer*, allo scadere del quale il server declassa n, nodo leader dal quale non ha ricevuto risposta in seguito a un messaggio di *CachePing*, da nodo Leader a nodo non Leader, scegliendo un nuovo Leader dalla cache.

Per il calcolo delle triangolazioni locali dei nodi viene effettuato il *Neighbor Test*(**??**): questo viene eseguito da un nodo n che viene a conoscenza di un nuovo nodo v. Il Neighbor test effettuato da n per un vicino candidato v è valido se il nodo v è un vicino di Delaunay di n. Le Definizioni 3.9 e 3.10 definiscono il *Clockwise Neighbor* e il *CounterClockwise Neighbor* di un nodo n rispetto a x.

Definizione 3.9 (Clockwise Neighbor)**.** Si dice *Clockwise Neighbor* $CW_x(n)$ di n rispetto a x il nodo b che soddisfa le seguenti proprietà:

1. forma il minore angolo in senso orario con il nodo x avendo n come pivot.

2. il minore angolo che forma è minore di 180 gradi.

Definizione 3.10 (CounterClockwise Neighbor)**.** Si dice *CounterClockwise Neighbor* $CW_x(n)$ di n rispetto a x il nodo b che soddisfa le seguenti proprietà:

1. forma il minore angolo in senso antiorario con il nodo x avendo n come pivot.

2. il minore angolo che forma è minore di 180 gradi.

Effettuare il Neighbor test su n per il nodo v, candidato a possibile vicino di Delaunay di n, significa verificare che il quadrilatero n, $CW_v(n)$, $CCW_v(n)$ e v con la diagonale che congiunge n e v, sia una triangolazione di Delaunay. Più in dettaglio, questo test equivale a verificare che la somma degli angoli in $CW_v(n)$ e $CCW_v(n)$ sia minore di 180 gradi. Se ciò avviene, allora il Neighbor Test per v è verificato, altrimenti non è verificato.
Per la memorizzazione dei vicini di Delaunay e dei nodi necessari all'esecuzione del Neighbor Test, ad ogni nodo $n \in V$ è associata una *tabella dei vicini* divisa in tre colonne: una colonna contiene l'identificatore del vicino v, mentre le altre due i nodi $CW_v(n)$ e $CCW_v(n)$. Esistono dei casi *speciali* in cui il Neighbor Test è verificato senza che valgano le condizioni descritte sopra, che rappresentano dei casi particolari nei quali il test non può essere applicato (Figura 3.5):

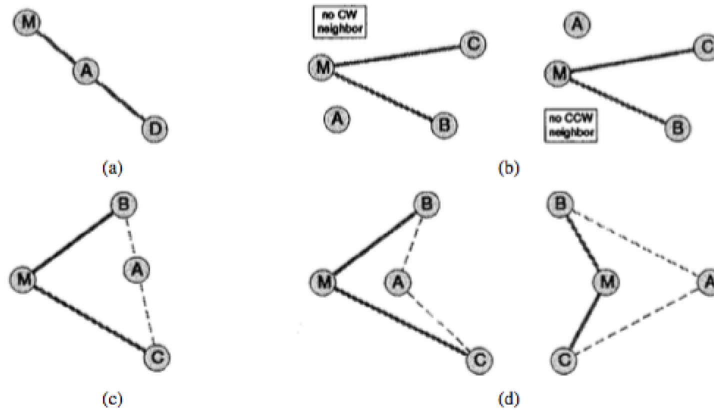

Figura 3.5: In Figura: Neighbor test in M per il nodo A, per i casi in cui la proprietà equiangolare non è applicabile poiché non è possibile definire un quadrilatero convesso dei nodi M, A, $CW_A(M)$, $CCW_A(M)$. La figura presenta scenari in cui A è aggiunto come vicino di Delaunay di M. Le linee solide rappresentano i vicini correnti di M, mentre le linee tratteggiate cono utilizzate per indicare il quadrilatero.

- Se N, X e B stanno sulla stessa linea, e B è vicino di X, N passa il test se è più vicino a X piuttosto che a B.

- Se N non ha CW e CCW neighbors rispetto a X, N passa il test.

- Se il quadrilatero formato da N, $CW_X(N)$, $CCW_X(N)$ e X è concavo o è un triangolo, N passa il test.

Tipi di nodi della rete

I nodi della rete possono essere di due tipi:

- i nodi *LEADER*, cioè nodi che hanno coordinate maggiori tra le coordinate di tutti i loro vicini conosciuti
- i nodi *NON LEADER*, cioè tutti gli altri nodi

Inoltre, i nodi possono assumere lo stato di *Candidate Neighbors* di altri nodi. Segue la definizione di *candidate neighbors.*

Definizione 3.11 (Candidate Neighbor)**.** Un nodo N è detto *Candidate Neighbor* di un nodo X se N non è nella tabella dei vicini di X e N è il *neighbor test* per N effettuato da X è verificato.

Ancora, i nodi possono essere *stabili* o *non stabili.* Un nodo N è stabile se nella sua tabella dei vicini ogni CW neighbor e CCW neighbor compare anche nella colonna dei vicini, altrimenti è detto non stabile.

Messaggi

In questa sezione sono definiti alcuni tipi di messaggi che garantiscono la correttezza dell'algoritmo. Alcuni di questi sono inviati tra nodi della rete, altri invece vengono inviati al server per notificare a questo la presenza di un determinato nodo nella rete o per effettuare la join del nodo. Distinguiamo i seguenti messaggi:

- i messaggi di *HelloNeighbor* e *HelloNotNeighbor.* Il primo viene inviato da ogni nodo, allo scadere del proprio Heartbeat, a tutti i nodi nella propria tabella dei vicini, e ai nodi candidate neighbors di cui si è venuti a conoscenza. Il messaggio di HelloNotNeighbor può essere inviato per tre motivi:
  - per fare aggiornare la tabella dei vicini del ricevente.
  - per dare qualche informazione in più al ricevente sul proprio vicinato, poiché tale messaggio contiene informazioni sui CW e CCW neighbors.
  - per risolvere problemi dovuti al fatto che più nodi hanno lo stesso indirizzo logico (cosa possibile poiché gli indirizzi logici vengono assegnati in fase di configurazione).

- il messaggio di *Goodbye* viene inviato da un nodo N nel momento in cui sta per effettuare la *Leave*. Dal momento in cui N invia tale messaggio, risponderà fino alla sua uscita con tale messaggio ad ogni nuovo messaggio che gli si presenta.

- il messaggio di *NewNode*, che viene inviato da un nodo N a un nodo X che, in fase di join, è appena venuto a conoscenza di quest'ultimo già presente nella rete. Questo messaggio può essere inviato da X già presente nella rete a un nodo più vicino di se stesso all'originario mittente del messaggio nel caso in cui N fallisca il neighbor test per X.

Algoritmo

L'algoritmo viene descritto in termini di due procedure differenti: una per la fase di *Join* e una per la fase di disconnessione volontaria dalla rete dei nodi.

La procedura di Join di un nodo n consiste nei seguenti punti:

- il nodo n, non ancora presente nella rete, contatta il server inviando un messaggio di *ServerRequest*, con lo scopo di venire a conoscenza di un nodo già presente nella rete. Il server cerca nella sua cache list, che contiene un insieme di nodi conosciuti, e risponde a n con un messaggio *ServerResponse* contenente un nodo x scelto casualmente nella sua cache list;

- il nodo n invia un messaggio *NewNode* al nodo x;

- il nodo x effettua il neighbor test per n. Se fallisce, inoltra il messaggio *NewNode* al prossimo nodo nella sua tabella dei vicini che sa essere più vicino a n, altrimenti invia un messaggio di *HelloNeighbor* a n. Tale messaggio conterrà i nodi $CCW_x(n)$ e $CW_x(n)$;

- il nodo n riceve il messaggio di *HelloNeighbor* da parte di x, venendo a conoscenza di due nuovi elementi che saranno candidati a essere vicini (*candidate neighbors*). Quindi, allo scadere del timer *HeartBeart*, n invia messaggi di *HelloNeighbor* a tutti i suoi vicini, per notificare loro che è ancora attivo nella rete, e al più vicino dei suoi candidate neighbors. I vicini e il candidate neighbor effettueranno quindi il neighbor test e risponderanno a n con un nuovo messaggio *HelloNeighbor*, il quale può contenere nuovi probabili candidate neighbors per n.

- se N non ha più candidate neighbors, il suo inserimento nella rete è terminato. Comunque, allo scadere dell'Heartbeat, n continua a inviare messaggi di $HelloNeighbor$ a tutti i suoi vicini.

La procedura di disconnessione volontaria di un nodo n consiste nei seguenti punti:

- n vuole uscire dalla rete, quindi invia messaggi di *Goodbye* a tutti i suoi vicini e al server. Da quel momento, n risponderà con messaggi di *Goodbye* a tutte le richieste che gli arriveranno.

- il server riceve il messaggio *Goodbye*, quindi rimuove n dalla sua cache.

- i vicini di n ricevono il messaggio *Goodbye*, quindi rimuovono n dalla loro tabella dei vicini. Comunque, n potrebbe essere stato salvato come CounterClockwise neighbor o come Clockwise neighbor di qualche nodo. Per definizione, n adesso è un candidate neighbor dei propri vicini, quindi allo scadere del loro Heartbeat essi invieranno messaggi di HelloNeighbor a n. Inoltre, essi inviano messaggi di *helloNeighbor* a tutti i loro vicini, in modo da aggiornare i CounterClockwise neighbors e i Clockwise neighbors di tutti i nodi della rete. (Non contengono più N poiché N è declassato a candidate neighbor).

- n riceve messaggi di *HelloNeighbor* dagli ex-vicini, quindi risponde loro con messaggi di *Goodbye*.

- la rete si auto-ripara grazie a messaggi di *HelloNeighbor* inviati da tutti i nodi allo scadere dei loro *Heathbeat*.

3.4.2 L'algoritmo dinamico distribuito in d dimensioni di M. Steiner et al.

In questa sezione sarà descritto l'algoritmo proposto da Steiner et al. in [33], un algoritmo distribuito per il calcolo della triangolazione di Delaunay in uno spazio d-dimensionale. La costruzione di una triangolazione di Delaunay in uno spazio d-dimensionale in maniera distribuita differisce dal caso a due dimensioni perché non è possibile definire i Clockwise Neighbor e CounterClockwise Neighbor di un nodo n rispetto a un altro nodo n. In dettaglio, Steiner et al. descrivono in [33] una procedura incrementale per l'inserzione e una per la cancellazione prima nel caso di $d = 2$, poi per $d = 3$ e infine generalizzando il tutto per ogni valore di d.
In questa sezione sono descritti due algoritmi:

1. l'algoritmo da applicare a nodi in uno spazio a due dimensioni, il quale fa uso del concetto di Clockwise Neighbor e CounterClockwise Neighbor;

2. l'algoritmo da applicare a nodi in uno spazio a tre dimensioni, il quale è sufficiente per descrivere il caso d-dimensionale.

Per ognuno di questi algoritmi, è descritta una procedura eseguita dai nodi in fase di join, una eseguita in fase di disconnessione volontaria dalla rete e una per la gestione dei fallimenti dei nodi nella rete.

Join di un nodo n in 2-space

Durante la fase di join di un nodo n in uno spazio a due dimensioni al tempo t, viene individuato nella triangolazione di Delaunay calcolata il triangolo che lo contiene, il quale viene diviso in 3 nuovi triangoli e, ricorsivamente, vengono controllati tutti i lati, che flippano nel caso in cui tale triangolazione non soddisfa la proprietà equiangolare (Sezione 1.2).

La procedura di Join dell'algoritmo prevede l'utilizzo dei seguenti messaggi:

- il messaggio *find-nearest*
- il messaggio *nearest*
- il messaggio *best*

Il messaggio *find-nearest* viene inviato dal nodo n per cercare il nodo più vicino a lui nella rete. il nodo n invia il messaggio a un nodo che sa essere partecipe nella rete, detto a, il quale dà inizio a un algoritmo di rgreedy routing per trovare il nodo più vicino a n. In dettaglio, a controlla se esiste un nodo v nella sua lista dei vicini di Delaunay tale che la distanza euclidea tra v e n è inferiore rispetto a quella tra a e n. In tal caso, a invia l'informazione di v a n utilizzando un messaggio *nearest*, altrimenti risponde a n con un messaggio *best*.

Algorithm 3 Ricezione di un messaggio di find-nearest da parte di un nodo A

```
1: procedure FIND NEAREST MESSAGE RECEIVED BY NODE A(Node N)
2:     Node nearest ← A
3:     for Every Node V Neighbor of A do
4:         if V is nearer than nearest then
5:             nearest ← V
6:     if nearer is equal to A then
7:         A sends a message best to N
8:     else
9:         A sends a message nearest to N containing nearest, N's successor and N's predecessor
```

Ricevuto il messaggio *best*, n viene a conoscenza del nodo più vicino a lui v, del predecessore di n secondo v e de successore di n secondo v. Tali nodi rappresentano i vertici del triangolo nel quale è inserito n. Ad ognuno di essi, il nodo n invia un messaggio *hello*, utilizzato per la connessione tra i nodi.
Alla ricezione di un messaggio *hello* da parte di n, il nodo a aggiorna la sua triangolazione locale tenendo conto di n. Una volta effettuato l'aggiornamento è possibile che qualche arco di cui a è un vertice subisca un'operazione di flip. In questo caso, il nodo a invia a n un messaggio *detech*, il quale conterrà l'informazione del nodo d che n dovrà aggiungere alla lista dei vicini. Infine, n si mette in contatto con d tramite un messaggio *hello*.

L'algoritmo procede ricorsivamente finché non ci sono più messaggi nella rete. L'algoritmo 4 descrive la ricezione di un messaggio *hello* da parte di un Nodo a.

Algorithm 4 Ricezione del messaggio Hello da parte di a

```
 1: procedure HELLO RECEIVED BY A(Node n)
 2:     Add n to neighbors
 3:     p_j ← successor(n)
 4:     p_{j-1} ← successor(p_j)
 5:     while n belongs to circumcicle of the triangle composed by a, p_{j-1}, p_j do
 6:         sends to n detect(p_{j-1})
 7:         remove p_j from neighbors
 8:         p_j ← p_{j-1}
 9:         p_{j-1} ← successor(p_j)
10:     p_j ← predecessor(n)
11:     p_{j+1} ← predecessor(p_j)
12:     while n belongs to circumcicle of the triangle composed by a, p_j, p_{j+1} do
13:         sends to n detect(p_{j+1})
14:         remove p_j from neighbors
15:         p_j ← p_{j+1}
16:         p_{j+1} ← predecessor(p_j)
```

Leave e failure di un nodo n in 2-space

La procedura di Leave definisce una serie di azioni che il nodo n deve compiere in fase di disconnessione volontaria dalla rete. In particolare, un nodo n che vuole disconnettersi deve:

1. informare i vicini di n $N(n)$ della loro uscita dalla rete;
2. calcolare nuovi vicini dei nodi in $N(n)$.

L'algoritmo prevede dei meccanismi per la gestione del fallimento di un nodo n. Purtroppo, questo è un caso ottimista, poiché è possibile che un nodo N fallisca senza aver la possibilità di inviare i nuovi collegamenti a un suo vicino V. Questo si risolve utilizzando la seguente proprietà delle triangolazioni:

Proposizione 3.4.2. *Sia $N(V)$ l'insieme dei nodi vicini a V in una triangolazione in uno spazio bidimensionale. Se $A \in N(B)$ e $B \in N(A)$, allora pred(B) e succ$(B) \in N(A)$ e viceversa.*

Quando un nodo V viene a conoscenza del fallimento di un suo vicino N, identifica il successore e il predecessore di N nella sua lista dei vicini, e invia loro un messaggio *lost*, contenente il nodo che ha subito il fault (V) e l'altro nodo a cui ha inviato il messaggio (il

precedessore al successore e viceversa). Predecessore, successore, il nodo che riceve il messaggio e il successore del successore costituiscono un quadrilatero sui quali triangoli verranno effettuati i test del cerchio vuoto, al fine di calcolare la triangolazione di Delaunay corretta.

Join di un nodo in 3-space

In uno spazio a tre dimensioni, il problema di costruire una triangolazione di Delaunay diventa quello di suddividere lo spazio tridimensionale in *tetraedri* non sovrapposti tra loro tale che la circumsfera passante per ogni tetraedro non contenga alcun altro punto dello spazio. Come accennato nella sezione introduttiva dell'algoritmo, in spazi con dimensioni maggiori di 2 non esiste il concetto di predecessore e successore di un nodo, quindi bisogna memorizzare per ogni nodo i tetraedri di appartenenza esplicitamente per colmare questa lacuna. Sia $T(n, t)$ l'insieme di tetraedri di appartenenza di n al tempo t, e sia $C(a, b, c, d)$ la circumsfera passante per i nodi a, b, c, d.

L'entrata di un nuovo nodo nella rete segue di pari passo il procedimento applicato nel caso di uno spazio bidimensionale: inizialmente, viene individuato il nodo v più vicino a n, che sta entrando nella rete; quindi, il tetraedro contenente n viene individuato da v e viene inviato a n. Il nodo n quindi divide il tetraedro ricevuto in 3 nuovi tetraedri. A differenza dell'algoritmo in uno spazio a due dimensioni, questa volta è necessario memorizzare interamente i tetraedri per potere conoscere il tetraedro di appartenenza di n.

Successivamente viene effettuato il test della *sfera vuota* ed eseguiti eventuali flipping di lati.

I messaggi utilizzati sono anche questa volta i messaggi di *find-nearest*, *nearest* e *best*, rispettivamente per trovare il nodo più vicino, per notificare un nodo più vicino e per comunicare di essere il nodo più vicino.

Definizione 3.12. Sia lo spazio tridimensionale diviso in due sottospazi tridimensionali da una faccia f di un tetraedro T. Definiamo la funzione booleana $same-side(f, e, z)$, la quale vale *true* se e solo se i nodi e e z appartengono allo stesso sottospazio diviso da f.

Definizione 3.13. Sia f una faccia di un tetraedro T. definiamo la funzione $opposite(f)$, la quale ritorna il nodo n non appartenente alla faccia f del tetraedro.

Per identificare se un tetraedro $T \in T(a, t)$ di a è il tetraedro che contiene il nodo n, viene valutata la seguente funzione:

Proposizione 3.4.3. $\forall f \in T \; same - side(f, opposite(f), Z) \; = \; true$

Il nodo a effettua un ciclo su tutti i suoi tetraedri per trovare quello che contiene n. Una volta trovato, lo invia a n attraverso un messaggio di *best*. n riceve il messaggio, quindi divide

il tetraedro in 3 parti e invia ai 4 nodi nel tetraedro originale un messaggio di *hello* contenente i nuovi tetraedri. A questo punto, il nodo A controlla i tetraedri ricevuti, effettuando dei test delle sfere vuote: se il tetraedro T_i non passa il test, allora viene individuato il nodo opposto a N rispetto a tale tetraedro e viene effettuata un'operazione di flipping. In questo caso, A notifica il nodo opposto a N della presenza di N con un messaggio *detect*.
Se invece il tetraedro T_i supera il test, allora questo viene inserito nell'insieme dei tetraedri di A. Uno pseudocodice relativo all'algoritmo è mostrato nell'Algoritmo 5.

Algorithm 5 Ricezione del messaggio Hello da parte di a in uno spazio tridimensionale

```
1: procedure HELLO RECEIVED BY A(Node z, T1, T2, T3)
2:     Q.put(T1)
3:     Q.put(T2)
4:     Q.put(T3)
5:     while Q ≠ ∅ do
6:         Ta ← Q.pop()
7:         face ← f ∈ Ta : opposite_Ta(f) = z
8:         Tb ← T ∈ T(a,t) : face ∈ T
9:         if z ∈ C(Tb) then
10:            Ti, Tj, Tk ← split(Ta, Tb)
11:            Q.put(Ti)
12:            Q.put(Tj)
13:            Q.put(Tk)
14:            e ← opposite_Tb(face)
15:            send detect(e, Ta) to z
16:            remove Tb from T(a,t)
17:        else
18:            insert T1 in T(a,t)
```

3.4.3 L'algoritmo incrementale distribuito di M. Ohnishi et al. per Virtual Collaborative Spaces

Ohishi et al. descrivono in [27] un algoritmo incrementale da utilizzare in *Virtual collaborative spaces*. Un Virtual collaborative space una rete in cui i nodi si scambiano una serie di messaggi. Tali nodi possono essere utenti veri e propri, avatar oppure dei bot appositamente costruiti.
A livello tecnico, assumiamo che tali nodi:

1. abbiano conoscenza della loro posizione geografica nella rete, rappresentata da coordinate (x, y);

2. conoscono il proprio identificativo logico;

3. hanno potenza di calcolo e sono autonomi;

4. hanno la possibilità di inviare messaggi nella rete;
5. hanno un raggio di visione limitato, quindi sono in grado di conoscere solo alcuni degli altri nodi.

Si assuma, inoltre, che in fase iniziale esista un grafo che colleghi i nodi ai suoi vicini. L'algoritmo propone la costruzione di una triangolazione di Delaunay utilizzando solamente l'informazione locale dei nodi, e propagando tale informazione a tutti i nodi nelle vicinanze. Per conservare l'informazione necessaria, ad ogni nodo vengono associate delle liste:

1. una lista di vicini, i quali rappresentano i vicini di un nodo nella rete;
2. una lista di vicini di Delaunay ($LDN(v)$), la quale viene utilizzata per tener traccia dei nodi collegati al nodo v tramite archi di Delaunay;
3. una lista di vicini non di Delaunay, i quali sono tutti i vicini che non sono di Delaunay.

Nella descrizione dell'algoritmo, distinguiamo 3 step logici:

1. l'inserzione di un nodo nella rete con triangolazione già costruita
2. il calcolo della triangolazione locale da parte di un nodo
3. il workflow dell'algoritmo vero e proprio.

Inserzione di un nodo

L'*inserzione* di un nodo nella rete avviene attraverso i seguenti passaggi:

- il nodo v_{n+1} viene piazzato nello spazio contenente altri n nodi.
- tra questi n nodi, v_{n+1} sceglie un nodo a caso, che chiameremo v_x
- finché v_x non è il nodo più vicino a v_{n+1}, v_x diventa il nodo più vicino a v_{n+1} tra i suoi vicini di Delaunay.
- una volta trovato v_x più vicino a v_{n+1}, viene controllato se v_{n+1} è incluso nei circumcerchi di tutti i triangoli di v_x o devi vicini di Delaunay di v_x. Se si, viene inviata l'informazione di v_{n+1} a questi nodi.
- modificare la lista di Vicini di Delaunay di v_x con l'informazione ottenuta da v_{n+1}.

Calcolo della triangolazione locale

Il calcolo della triangolazione locale per un nodo n data la conoscenza di un insieme di vicini procede nel seguente modo: inizialmente, i vicini in $N(n)$ vengono ordinati in ordine orario prendendo come angolo 0 di riferimento la semiretta uscente da n e andante verso nord.
Poi, vengono presi a due a due i nodi di tale insieme ordinato per formare un triangolo con n. Viene quindi preso il nodo successivo dall'insieme di vicini ordinati e verificata la proprietà del cerchio vuoto sul quadrilatero formato da n, i due punti presi precedentemente e il nodo successivo. Se vale, si continua procedendo finché l'insieme ordinato non viene verificato interamente. Se non vale, si elimina la connessione tra n e il secondo nodo dei due punti presi precedentemente (poiché si è verificato un flipping). L'eliminazione del nodo avviene ponendolo nella lista dei vicini non di Delaunay. In dettaglio:

1. il nodo n ordina in ordine orario i nodi nella sua lista di vicini $N(n)$.
2. questo prende il nodo *top* da $N(n)$ e lo pone in fondo a $LDN(n)$.
3. se $LDN(n)$ contiene più di tre nodi (siano questi nodi $l_n, l_{n-1}el_{n-2}$) forma un triangolo con l_n, l_{n-1} e n e disegna il circumcerchio.
   (a) controlla se l'arco nl_{n-2} interseca l'arco l_nl_{n-1}
   (b) controlla se il circumcerchio formato contiene $l_n - 2$.
   (c) se si verifica una delle due condizioni sopra, allora sposta l_{n-1} in $NNL(n)$ (lista dei non Delaunay Neighbors).
4. ripeti tale procedura finché c'è ancora informazione all'interno dell'insieme $N(n)$ e finché non si è arrivati in fondo alla lista dei $LDN(n)$.

Tale procedura è molto semplice e garantisce che la visione locale di ogni nodo abbia una triangolazione di Delaunay corretta. Purtroppo, questo non basta per la costruzione di una triangolazione di Delaunay globale corretta, poiché possono verificarsi delle inconsistenze date dalla mancata conoscenza di alcuni nodi poiché fuori dalla zona interessata di un nodo n. Inoltre, alcune triangolazioni potrebbero sovrapporti ad altre, non rispettando il vincolo di planarità della triangolazione in globale.
Per questi motivi, viene utilizzato un meccanismo di scambio delle informazioni contenute negli insiemi $LDN(n)$ e $NDN(n)$ tra nodi specifici.

L'algoritmo

L'algoritmo distribuito su reti P2P segue i seguenti step:

	Liebeherr et el. [24]	Steiner et al.	Ohishi et al. [27]
Completamente Distribuito	No	No	No
Utilizzo di timers	Si	Non definito	No
Supporto d-dimensioni	No	Si	No
Supporto concorrenza join	No	No	No
Supporto churn	Si	Si	No

Tabella 3.3: Confronto tra algoritmi per Reti distribuite e P2P

1. la lista di informazioni del nodo n è bloccata per evitare che un altro nodo ci acceda in tale momento. Tale passaggio è fondamentale per evitare l'inserimento di nodi in parallelo, il quale non è supportato dall'algoritmo.
2. il nodo n calcola la propria lista $LDN(n)$ con il procedimento spiegato in questo paragrafo.
3. il nodo n invia l'informazione l'informazione n a tutti i suoi vicini di Delaunay, in maniera tale da notificarli della propria presenza nella rete.
4. il nodo n invia l'informazione presente in $NDN(n)$ a tutti i nodi più vicini ai suoi vicini di Delaunay, e cancella questa informazione.
5. il nodo n invia l'informazione presente in $LDN(n)$ a tutti i suoi vicini di Delaunay, in maniera tale da notificarli della propria vista locale. Questi, costruiscono dei triangoli con n e con i vicini di Delaunay di n.
6. sblocca la lista di informazioni di n.

3.4.4 Conclusioni

Nelle precedenti sezioni sono stati analizzati 3 algoritmi distribuiti per la costruzione della triangolazione di Delaunay. L'algoritmo di Liebeherr et al. è un algoritmo distribuito per la costruzione della triangolazione di Delaunay in uno spazio a due dimensioni che fa uso dell'utilizzo di messaggi di sincronizzazione tra i nodi per poter realizzare una versione distribuita del Flip Algorithm. I messaggi di sincronizzazione tra questi sono inviati tramite l'utilizzo di appositi timers e il tutto viene coordinato con l'ausilio di un'unità centralizzata.
L'algoritmo di Steiner et al. supporta gli spazi a d-dimensioni, con $d > 2$, e prevede una serie di procedure per la gestione della fase di Join, della fase di Leave e del fallimento dei nodi. Questo realizza una versione distribuita del *Flip Algorithm.*

La Tabella 3.3 mostra le principali differenze tra gli algoritmi analizzati. Dall'analisi è risultato che tutti i protocolli non sono in grado di gestire join di nodi concorrenti nella rete,

a causa del fatto che tutti utilizzano il greedy routing per trovare il nodo più vicino al nodo che sta effettuando la join.
Liebeherr utilizza timers e un sistema centralizzato per la distinzione delle tipologie di nodi nella rete. L'algoritmo di Steiner et al., invece, non utilizza esplicitamente i timers ma descrive superficialmente una procedura per il rilevamento dei fallimenti dei nodi nella rete, quindi è supposto che una qualche forma di temporizzazione sia presente per effettuare il controllo di tale fallimento da parte dei nodi vicini. L'algoritmo di Ohnishi et al. non utilizza timers, ma non ha il supporto al churn di rete.

3.5 Algoritmi distribuiti di supporto a problemi di Copertura di Aree

In questa sezione sono presentati gli algoritmi orientati alla copertura di aree geografiche. Un algoritmo di supporto alla copertura di aree deve garantire che una determinata area sia completamente controllata da una serie di controllori hardware di questa, per esempio da un insieme di sensori.
La copertura di aree trova principalmente applicazione nel campo del monitoraggio geografico di un territorio. Gli algoritmi presentati in questa sezione sono *DT-Score* [43] e *Improved Delaunay Triangulation* [40].

3.5.1 Delaunay Triangulation Score

Delaunay Triangulation Score [43] (DT-Score) è stato realizzato da Chun-Hsien Wu, Kuo-Chuan Lee e Yeh-Ching Chung con lo scopo di risolvere problemi di *area coverage* in una rete con sensori wireless. Si definisce con il termine di *area coverage* il problema di posizionare un insieme di sensori in modo da coprire quanto più tutto lo spazio disponibile. DT-Score ha lo scopo di massimizzare tale copertura utilizzando un insieme di sensori nel miglior modo possibile, anche in presenza di ostacoli. Per quanto l'algoritmo sia interessante dal punto di vista scientifico, sarà descritto molto brevemente poiché la descrizione dettagliata dell'algoritmo non rientra nei nostri scopi, in quanto lo scopo della tesi non è quello di analizzare algoritmi per la costruzione della triangolazione di Delaunay in presenza di ostacoli, ed inoltre questo algoritmo è centralizzato.

L'algoritmo prevede due fasi:

1. una fase per eliminare aree non coperte nelle vicinanze dei confini della rete e degli ostacoli;

2. una fase di posizionamento dei sensori wireless.

La prima fase di generazione dei nodi di contorno si divide a sua volta in due fasi:

1. inizialmente, viene letto un file di configurazione al fine di caricare nella rete eventuali ostacoli e una serie di sensori di default. Nel file di configurazione sono presenti tutte le informazioni necessarie, quindi raggio massimo di un insieme di sensori da posizionare nell'area, detti *sensori di default*, e posizionamento dei sensori e degli ostacoli.

2. successivamente, viene utilizzato un algoritmo per determinare i punti di contorno dello spazio in cui andremo a piazzare i sensori e degli ostacoli. Tale algoritmo non verrà analizzato, poiché fuori dai nostri scopi.

La seconda fase, di calcolo delle posizioni dei sensori, si divide in tre parti:

1. la fase di calcolo delle posizioni candidate: durante questa fase, la quale avviene in maniera centralizzata e offline, viene calcolata una triangolazione di Delaunay sui punti di contorno generati al passo precedente. Dopo questo calcolo, la rete presenterà una serie di triangoli i cui circumcerchi siano vuoti (non contengono nessun punto, poiché altrimenti la triangolazione non sarebbe di Delaunay). Tra questi, quelli che non contengono nessun sensore sono scelti e il loro centro viene salvato all'interno di un vettore di posizioni candidate per i sensori.

2. la fase di scoring, un cui viene assegnato un punteggio ad ogni posizione candidata del vettore calcolato al passo precedente, poiché solo un insieme di posizioni prestabilito verrà utilizzato per i sensori da aggiungere

3. la fase di aggiunta sensori, in cui banalmente, a seconda del punteggio ottenuto nella fase precedente, un sensore viene aggiunto alla posizione *prossima posizione candidate* nel vettore di sensori.

3.5.2 Improved Delaunay Triangulation

Chinh T. Vu e Yingshu Li descrivono in [40] un algoritmo basato sulle triangolazioni di Delaunay da utilizzare per problemi di area coverage in reti di sensori wireless. Essi prendono spunto da un algoritmo già analizzato e descritto in [41] e, partendo da questo, ne realizzano una versione migliorata che non soffrisse del *boundary effect issue*, definito come il problema del non riuscire a coprire le aree ai confini del piano considerato.

A differenza dell'algoritmo presentato nel paragrafo precedente, Vu et al. risolvono il boundary effect issue assegnando un raggio adeguato a ognuno dei sensori posizionati nella rete, e trovando la posizione ottima nella rete per tali sensori.

Nella prossima sezione, viene descritto brevemente l'algoritmo originale di [41] a cui Vu et al. hanno fatto riferimento per la realizzazione di Improved Delaunay Triangulation.

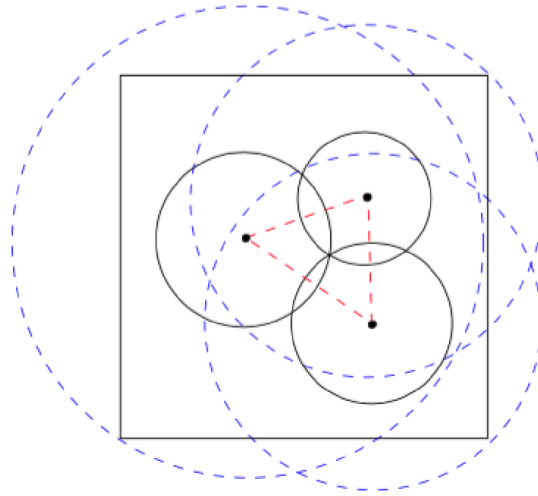

Figura 3.6: Esempio di boundary effect issue. Il triangolo (in rosso) rappresenta la triangolazione di Delaunay costruita sui tre nodi. I cerchi tratteggiati rappresentano il massimo raggio d'azione possibile per i sensori, mentre i cerchi in linea solida il raggio calcolato dall'algoritmo originale, i quali danno luogo all'errore

L'algoritmo originale

L'algoritmo descritto in [41] fornisce un insieme di direttive su come aggiornare correttamente il raggio dei sensori tenendo conto della loro energia residua e della propria posizione geografica nella rete. Obiettivo dell'algoritmo è quello di stabilire dei raggi per i sensori tali che i triangoli della triangolazione di Delaunay *approssimata* costruita su questi sia totalmente coperta. Nella costruzione di tale triangolazione ogni nodo della rete tiene conto solamente dei sensori vicini raggiungibili con un hop.

La triangolazione costruita è approssimata perché si suppone che i nodi possano comunicare solamente con un sottoinsieme di nodi entro un raggio limitato da questi. Non si suppone, inoltre, l'esistenza di un sistema centralizzato per la gestione dei sensori.
L'algoritmo è composto da due fasi:

1. la fase di costruzione della triangolazione approssimata, nella quale ogni nodo costruisce la propria cella di Voronoi utilizzando le bisettrici degli archi che li collegano con i propri vicini immediati (distanti un hop), e nella quale ogni nodo usa la cella di Voronoi calcolata per ricavare la triangolazione di Delaunay corrispondente.

2. la fase di misura del raggio d'azione, la quale consiste nel calcolare il raggio di azione da assegnare ai sensori, il quale dipenderà dalla distanza del sensore dal centroide di ogni triangolo, il quale dipende anche dall'energia residua dei sensori vertici del triangolo, e dal massimo raggio d'azione possibile per il sensore.

Utilizzando questo algoritmo è però possibile che, se i sensori hanno una posizione distante dai confini della rete, alcune aree ai confini rimangano non coperte (Figura 3.6).

Improved Delaunay Triangulation

Vu et al. [40] riconsiderano il calcolo dei raggi d'azione da assegnare ai sensori nell'algoritmo originale al fine di coprire tutti gli spazi dell'area in input, in modo da risolvere il boundary effect issue.

Il raggio di ogni sensore può essere diviso in modo che ognuna delle partizioni venga coperta dallo stesso insieme di vicini. Si definisce col nome di *perimeter coverage level di una partizione* p il numero di vicini nv che coprono la partizione p. In particolare, si dice che il *perimeter coverage level* di un nodo n è uguale a k se k è il minimo perimeter coverage level delle partizioni del raggio d'azione di n. Se tutti i nodi della rete N hanno *perimeter coverage level* $= k$, allora la rete si dice *k-perimeter covered.* Nella rete che andremo a considerate, tutti i nodi hanno perimeter coverage level uguale a 1. Prima della descrizione dell'algoritmo, si considerino le seguenti definizioni:

Definizione 3.14. Si dice *porzione non coperta* una porzione del raggio di un sensore che non viene coperto da nessuno dei suoi vicini

Definizione 3.15. Si dice *nodo non coperto* un nodo che non è completamente coperto dal raggio dei propri vicini

Definizione 3.16. Si dice *vicino utile* per un nodo n un vicino di un nodo n che, utilizzando il massimo raggio assegnabile, è in grado di coprire una porzione non coperta di n.

Come l'algoritmo originale, l'algoritmo migliorato consiste in due fasi: una per la costruzione della triangolazione di Delaunay approssimata e una per il calcolo del raggio d'azione da assegnare ai sensori.
Una volta costruita la triangolazione di Delaunay approssimata con la stessa procedura descritta per l'algoritmo originale, lo stato della rete prevede che un nodo n possa essere includo in una serie di triangoli.
Per ognuno di questi triangoli, viene calcolato il centroide effettuando una media pesata sulle coordinate geografiche dei sensori, utilizzando l'energia residua come peso. Il raggio d'azione dei sensori viene quindi aggiornato alla distanza tra il sensore e il centroide pesato appena calcolato. Se il nodo che sta calcolando il raggio d'azione è incluso in più di un triangolo, il raggio d'azione scelto è il massimo tra i centroidi pesati calcolati sui triangoli.
Il boundary effect issue viene risolto definendo una priorità tra i nodi della rete, in modo che i nodi con potenza maggiore possano calcolare il proprio raggio prima di nodi con potenza inferiore.

Oltre alla priorità, viene definito controllato se un nodo è 1-perimeter covered, e a seconda della priorità e questo stato vengono eseguite le seguenti azioni:

- il nodo è 1-perimeter covered, quindi si mette in attesa di eventuali nodi vicini che non lo sono e per cui tale nodo può essere un vicino utile
- il nodo non è 1-perimeter covered e ci sono ancora nodi prioritari che non hanno eseguito il test: in questo caso, il nodo aspetta che eventuali vicini non 1-perimeter covered aumentino il proprio raggio, in modo da coprire aree non coperte per lui
- il nodo non è 1-perimeter covered ed è il suo turno: in questo caso il nodo aumenta il proprio raggio al massimo possibile, e copre la porzione non coperta.
- il nodo non è 1-perimeter covered, è il tuo turno, ma aumentando il raggio non riesce a coprire la porzione non coperta: in questo caso, il nodo ricerca suoi vicini utili, invitandoli ad aumentare il raggio al massimo possibile

Vale infine il seguente teorema di correttezza per l'algoritmo IDT:

Teorema 3.5.1. *Se l'area può essere coperta completamente dalla rete a cui i sensori è assegnato il massimo raggio possibile, l'algoritmo IDT assicura che l'area venga coperta completamente.*

Una descrizione in pseudocodice è presentata dall'Algoritmo 6.

Algorithm 6 Improved Delaunay Triangulation eseguito dal sensore s

1: *Raccogli informazioni sui vicini locali di s*
2: *Calcolo della triangolazione di Delaunay approssimata*
3: **for** *Triangle t con nodi s, t, v nella triangolazione* **do**
4: $m_x = \frac{E(s)\times s_x + E(t)\times t_x + E(v)\times v_x}{E(s)+E(t)+E(v)}$
5: $m_y = \frac{E(s)\times s_y + E(t)\times t_y + E(v)\times v_y}{E(s)+E(t)+E(v)}$
6: $s.R = \max s.R, sm$
7: **if** *s ha priorità maggiore tra i suoi vicini e s non è 1-perimeter covered* **then**
8: *s aumenta il raggio al massimo possibile*
9: *s chiede ai vicini di aumentare il loro raggio al massimo possibile*

Capitolo 4

New ACE: un nuovo protocollo per la Delaunay distribuita

L'obiettivo di questo capitolo è quello di effettuare un'analisi dettagliata di alcuni algoritmi distribuiti per la costruzione della triangolazione di Delaunay al fine di poter effettuare dei confronti tra questi per valutarne le performance e le possibili problematiche.
Nel Capitolo 3 è stata presentata una rassegna dei principali algoritmi distribuiti per la costruzione della triangolazione di Delaunay. Gli algoritmi descritti sono stati divisi per costruzione e per applicazione:

- algoritmi localizzati orientati ad applicazioni alle reti di sensori wireless, perché la Delaunay è costruibile mediante una visione localizzata;
- algoritmi orientati ad applicazioni relative ad Ambienti Virtuali distribuiti;
- algoritmi orientati al routing geografico;
- algoritmi orientati a sistemi distribuiti e P2P.

Nella scelta degli algoritmi da confrontare è stato deciso di prendere in considerazione solamente algoritmi generali per la costruzione di una triangolazione di Delaunay in ambiente distribuito, in particolare per il vasto insieme di campi in cui questa struttura trova applicazione. L'analisi di algoritmi localizzati per la triangolazione di Delaunay oppure di algoritmi orientati alla copertura di aree avrebbe limitato l'applicazione della struttura costruita alle reti di sensori. Tra gli algoritmi che possono essere considerati di ambito generale, presentati nella rassegna del Capitolo 3, l'algoritmo di Liebeherr et al. è analizzato e spiegato in maniera chiara e formale [24], mettendo in evidenza anche i casi particolari in cui la proprietà equiangolare non può essere verificata e tutti i messaggi utilizzati, al contrario dell'algoritmo di Steiner et al. [33] e dell'algoritmo di Ohnishi et al. [27], che non sono descritti in dettaglio. D'altro canto, l'algoritmo di Liebeherr richiede l'implementazione di una parte centralizzata

per la gestione dei nodi leader, della cache e dei meccanismi di bootstrap, e una serie di eventi temporizzati.
Per questo motivo, la scelta si è focalizzata su due algoritmi, la cui descrizione non è stata inserita nella rassegna, ma che riteniamo particolarmente ineteressanti:

1. *GoDel* (Gossip Delaunay) [5], un protocollo distribuito che utilizza algoritmi di gossip per il riempimento della vista locale dei nodi, la quale viene poi utilizzata per il calcolo dei vicini locali di Delaunay dei nodi della rete;

2. *ACE* (Accuracy Correctness Efficiency) [21], un protocollo distribuito che basa la costruzione della vista locale sull'approccio *Candidate Set*, il quale prevede la memorizzazione di una lista dei nodi conosciuti in fase di join per il calcolo dei vicini locali di Delaunay.

I due protocolli presentano delle caratteristiche che li distinguono da quelli presentati nella rassegna. GoDel è l'unico algoritmo distribuito per la costruzione della triangolazione di Delaunay che fa uso di algoritmi di gossip per riempire la vista locale dei nodi. Gli algoritmi di gossip sono una classi di algoritmi che permettono la diffusione di una vasta quantità di informazione nella rete sfruttando lo scambio delle viste locale di nodi scelti in modo casuale. GoDel utilizza gli algoritmi di gossip per effettuare degli scambi di conoscenza tra nodi casuali della rete in modo da permettere ad ogni nodo di conoscere i propri vicini di Delaunay. Altra caratteristica di GoDel è che non richiede meccanismi temporizzati per la riparazione della rete, poiché gli algoritmi di gossip fanno si che la rete si auto-stabilizzi in modo continuo. Ultima caratteristica interessante è la veloce convergenza, data sempre dagli algoritmi di gossip: GoDel calcola una triangolazione di Delaunay corretta in un numero basso di iterazioni. ACE prevede la costruzione della triangolazione di Delaunay utilizzando l'approccio con Candidate Set: in fase di join, ogni nodo contatta un insieme di nodi della rete locati nelle vicinanze della posizione del nuovo nodo al fine di riempire la sua vista locale.

Come GoDel, ACE riempe la vista locale dei nodi utilizzando uno scambio di messaggi, ma, a differenza di questo, lo scambio di messaggi avviene con un insieme di nodi nelle vicinanze: in questo modo, la vista locale dei nodi viene riempita con potenziali vicini di Delaunay e il numero di messaggi per convergere a una triangolazione corretta è minimo. La scelta di ACE è motivata dal fatto che, come GoDel, è frutto di una pubblicazione recente (2008) e dal fatto che il numero di messaggi per la costruzione della triangolazione è basso grazie ad alcune ottimizzazioni introdotte nel protocollo. Inoltre, ACE costruisce una triangolazione completa, a differenza di altri algoritmi analizzati, e si differenzia da altri algoritmi perché:

1. utilizza un solo servizio temporizzato, a differenza dell'algoritmo di Liebeherr et al. [24];

2. è descritto molto bene in [21], non lasciando grossi margini di interpretazione per il lettore, a differenza dell'algoritmo di Ohnishi[27] e dell'algoritmo di Steiner [33].

Il capitolo è strutturato come segue: la sezione 4.1 presenta una descrizione in dettaglio del protocollo GoDel. La sezione 4.2 fornisce una descrizione dettagliata di ACE, con particolare attenzione a tutte le sue componenti. La sezione 4.3 descrive alcune osservazioni ricavate da un'analisi dettagliata dei Teoremi e i Lemmi di correttezza di ACE, la dimostrazione della non completa correttezza del quale ha portato alla realizzazione di un nuovo protocollo, chiamato New ACE, il quale è descritto nella sezione 4.4.

4.1 Il protocollo GoDel

In questa sezione è presente una descrizione del protocollo GoDel, descritto in dettaglio in [5] e realizzato da Ranieri Baraglia, Patrizio Dazzi, Barbara Guidi, Laura Ricci.
L'originalità presentata da GoDel sta nell'utilizzo di *algoritmi di Gossip*, utilizzati per la ricerca di nuovi nodi all'interno della rete. GoDel utilizza tali algoritmi per fornire ai nodi la conoscenza necessaria per il calcolo delle triangolazioni di Delaunay locali.
In letteratura, i *protocolli epidemici o di Gossip* sono una classe di algoritmi utilizzata per diffondere l'informazione in modo epidemico all'interno della rete. Il nome di tale classe di algoritmi prende spunto dalla vita reale, in particolare dalla diffusione di un pettegolezzo. Applicazioni molto comuni per gli algoritmi di gossip sono il calcolo di funzioni di aggregazioni e statistiche nelle reti, poiché si riesce ad ottenere molto velocemente la conoscenza necessaria per effettuare tale calcolo spargendo messaggi all'interno di tali reti.
Baraglia et al. [5] utilizzano la proprietà degli algoritmi di Gossip per poter ottenere l'informazione relativa ai nodi presenti nella rete, mantenendo solo una vista locale limitata.
GoDel è composto da due livelli:

1. il livello superiore, il quale sarà riferito come *livello Godel*, rappresenta il livello in cui viene costruita la triangolazione di Delaunay distribuita, utilizzando le informazioni fornite dal livello di Gossip;

2. il livello di gossip, il quale utilizza il protocollo *Cyclon* [39], per lo scambio di viste tra i nodi.

Cyclon implementa un servizio di random peer-sampling ed è utilizzato per ottenere la conoscenza globale della rete in un numero limitato di passi. In questo livello, la comunicazione tra i nodi è stabilita per lo scambio delle viste locali di questi. La conoscenza acquisita tramite Cyclon viene poi utilizzata dal livello GoDel per costruire la triangolazione di Delaunay distribuita.

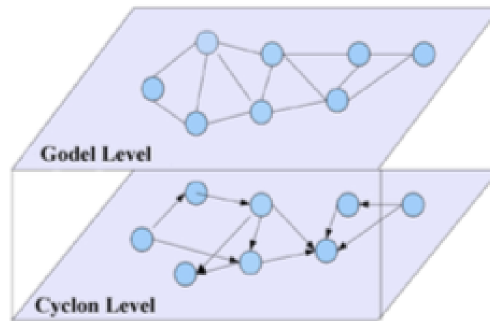

Figura 4.1: Il livello GoDel e il livello Cyclon che compongono il protocollo GoDel

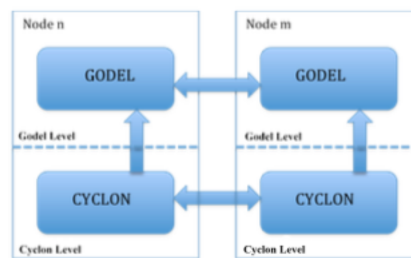

Figura 4.2: Interazioni tra i nodi a livello GoDel e livello Cyclon

4.1.1 Il livello GoDel

Lo scopo del livello GoDel è quello di costruire una overlay basata sulla triangolazione di Delaunay distribuita utilizzando l'informazione acquisita da Cyclon. Quando un nodo n viene a conoscenza di un nuovo nodo m, controlla se l'arco nm è un arco di Delaunay considerando tutti i vicini di Delaunay $DelN(n,t)$ di n al tempo t: se tale arco è di Delaunay, allora n inserisce m nella sua vista locale, altrimenti m viene scartato. Ovviamente, l'inserimento di m può causare il flip di qualche lato nella triangolazione.
L'algoritmo assicura una convergenza verso una triangolazione di Delaunay corretta per ogni nodo della rete, e che tale triangolazione è corretta se confrontata con la triangolazione globale calcolata sull'intero insieme di nodi della rete V.
Di seguito, viene data una definizione di correttezza per la triangolazione di Delaunay distribuita.

Definizione 4.1. Una *Triangolazione di Delaunay Distribuita* $DDT(V)$ di un insieme di nodi V al tempo t è definita come l'insieme di coppie $\{< u, DelN(u,t) > | u \in V\}$.

Definizione 4.2. Dato un insieme di nodi V, $DDT(V)$ al tempo t è corretta se e solo se per ogni coppia di nodi u e v di V tale che $u \in DelN(v,t)$ e $v \in DelN(u,t)$, allora l'arco

$uv \in DT(V)$, dove $DT(V)$ è la triangolazione di Delaunay sui nodi V.

Di grande importanza è il seguente teorema:

Teorema 4.1.1. *Si consideri $DT(V)$, la triangolazione di Delaunay globale sulla rete composta dai nodi V, e si consideri un sottoinsieme C di V. Allora, se $u \in C$ e $v \in C$ sono vicini di Delaunay in $DT(V)$, sono vicini di Delaunay anche in $DT(C)$.*

Il teorema dice che se due nodi sono vicini di Delaunay nella triangolazione globale $DT(V)$, allora lo saranno anche nelle triangolazioni locali, quindi quando un nodo n viene a conoscenza di un suo vicino di Delaunay m questo sarà inserito nella sua lista di vicini di Delaunay e vi rimarrà fino a quando sarà online.
Al contrario, se un nodo n viene a conoscenza di un nodo k che non è vicino di Delaunay di n in $DT(V)$, è possibile che k diventi suo vicino ma è garantito che sarà rimosso non appena n verrà a conoscenza di nuovi vicini di Delaunay in $DT(V)$, che attraverso il Test del Cerchio Vuoto permetteranno di eliminare k.

Teorema 4.1.2. *Si consideri un insieme di nodi C tale che l'insieme di vicini di Delaunay di un nodo n in $DT(V)$ sia incluso in C. Allora l'insieme di vicini di n in $DT(C)$ è uguale a l'insieme di vicini di Delaunay di n in $DT(V)$.*

Il teorema sta alla base del funzionamento di GoDel, perché mostra che quando un nodo viene a conoscenza di un suo vicino di Delaunay, questo viene individuato ed inserito nella vista indipendentemente dai vicini attuali.

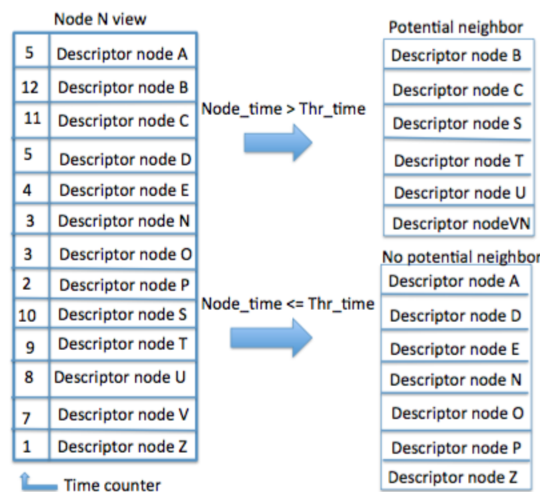

Figura 4.3: La vista locale di un nodo in GoDel

4.1.2 L'algoritmo

Ogni nodo in GoDel mantiene una lista di *descrittori di nodi* di cui è venuto a conoscenza, utilizzata per memorizzare la posizione geografica nella rete di tali nodi e le informazioni necessarie per il calcolo della triangolazione.
A livello Cyclon, ad ogni ciclo di gossip, ogni nodo sceglie un nodo dalla sua vista in modo casuale ed esegue uno scambio di vista al fine di aumentare la conoscenza locale.
A livello GoDel, ad ogni ciclo di gossip, viene utilizzato un sistema di ranking per la scelta dei nodi col quale effettuare lo scambio di informazioni. In dettaglio, ogni nodo assegna a ogni nodo nella sua vista locale un punteggio per determinare i nodi presenti da più tempo. Questo variabile viene incrementata ad ogni ciclo di gossip, in modo da stabilire un ordinamento tra i nodi conosciuti. Inoltre, viene utilizzata una *threshold* per la suddivisione della conoscenza in due gruppi distinti:

1. il primo gruppo contiene i nodi che stanno nella vista locale da più tempo, la quale memorizza i *vicini potenziali*;

2. il secondo gruppo contiene i nodi che sono presenti da meno tempo e viene chiamato *vicini non potenziali.*

Viene quindi utilizzata la seguente euristica: *più è il tempo che un nodo è stato in una vista locale, più è probabile che i due nodi siano vicini nella triangolazione di Delaunay globale.*

Ad ogni ciclo di gossip un nodo viene scelto dalla lista dei vicini potenziali con probabilità P_1, e dalla lista dei vicini non potenziali con probabilità P_2, con $P_1 >> P_2$ perché i vicini hanno la possibilità di inviare molte più informazioni necessarie al nodo, poiché la triangolazione di Delaunay è un problema locale.

Ad ogni ciclo di gossip vengono selezionati tutti i nodi a livello di Cyclon e un sotto nella vista a livello di GoDel: ogni nodo che è collegato al nodo n da un arco di Delaunay diventa un vicino di Delaunay. Il nodo n effettua il *neighbor test* per sapere se un nodo ricevuto dal livello GoDel o Cyclon è un vicino di Delaunay. A tal proposito, il nodo n mantiene i suoi vicini di Delaunay in una lista ordinata per angoli, considerando il nodo n come l'origine di un piano cartesiano e calcolando gli angoli seguendo l'ordine antiorario.

Per effettuare il neighbor test viene considerato il quadrilatero formato dal nodo n che sta aggiungendo il nuovo vicino e, il nuovo vicino, il successore e il predecessore del nuovo vicino. Viene quindi verificato che l'arco ne sia di Delaunay: se non lo è, il nodo e viene scartato, altrimenti e diventa un vicino di n e il lato tra il predecessore e il successore effettua un operazione di flip.

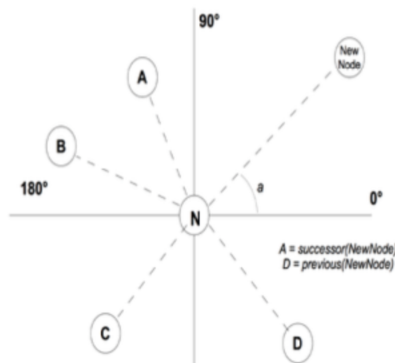

Figura 4.4: Ordinamento dei vicini del nodo N in GoDel

4.2 Il protocollo ACE

In questa sezione è descritto il protocollo ACE (Accuracy, Correctness and Efficiency), presentato da D. Lee e S. S. Lam in [21]. Questo protocollo è stato analizzato in questa tesi e ne è stata implementata una nuova versione che tiene conto di alcune anomalie riscontrate nell'analisi del protocollo. Come già anticipato nella parte introduttiva di questo capitolo, l'approccio utilizzato da ACE per il riempimento delle viste locali dei nodi è l'approccio Candidate Set, il quale consiste nel riempire una vista locale della rete con nodi conosciuti tramite uno scambio di messaggi tra nodi nelle vicinanze e nell'utilizzare tale insieme per il calcolo dei vicini di Delaunay. In dettaglio, l'insieme dei nodi conosciuti riempito mediante questo scambio di messaggi è detto *Candidate Set*, e fa parte della vista locale, insieme ad altre due liste: una per la memorizzazione dei vicini di Delaunay dei nodi, detto *Neighbor Set*, e uno per la memorizzazione dei nodi a cui si è inviata una qualche richiesta in passato, detto *Neighbor Queried Set*, utilizzata dal protocollo per minimizzare il numero di messaggi nella rete.
Nel seguito, indicheremo con $C(n)$ l'insieme *Candidate Set* del nodo n, con $N(n)$ l'insieme Neighbor Set di n e con $N^{queried}(n)$ l'insieme Neighbor Queried Set di n.
Al fine di facilitare la lettura, si consideri la Tabella 4.2 per i riferimenti alla notazione utilizzata.

ACE è composto da 4 sotto-protocolli:

1. un protocollo per la gestione della *join* dei nodi nella rete;
2. un protocollo per la gestione della *uscita* dei nodi dalla rete;
3. un protocollo per la gestione dei *fallimenti*;
4. un protocollo per la *manutenzione* della rete.

Notazione	Descrizione
$C(n)$	Candidate Set di n
$N(n)$	Neighbor Set di n
$N^{queried}(n)$	Neighbor Queried Set di n
$DT(C(n))$	Triangolazione di Delaunay locale di n calcolata in $C(n)$
$N_v(n)$	Vicini di v in $DT(C(n))$
$N_{v,n}^{mutual}(n)$	Vicini comuni a v e n in $DT(C(n))$

Tabella 4.1: Tabella di notazione

In base a quanto descritto in [21], la suite di protocolli è provata essere corretta per join, leave o eventi di fallimento sequenziali. Per la gestione di eventi concorrenti, viene utilizzato il protocollo di manutenzione della rete, il quale è parte integrante di ACE e ha lo scopo di ricercare nuovi nodi nelle vicinanze e di correggere eventuali inconsistenze dovute a leave e fallimenti concorrenti.

Il funzionamento del protocollo di join è il seguente: in fase iniziale, il nodo n inizializza i suoi insiemi Candidate Set $C(n) = \{n\}$, Neighbors Set $N(n) = \emptyset$ e Neighbors Queried Set $N^{queried}(n) = \emptyset$. $C(n)$ è inizializzato con il nodo corrente perché è necessario che questo sia sempre presente per il corretto calcolo dei vicini di Delaunay di n. Successivamente, il nodo n, in fase di Join, contatta un nodo a che partecipa nella rete, il quale fa partire un greedy routing per la ricerca del nodo c più vicino a n nella rete secondo la metrica euclidea. Il nodo c invia al nodo n un messaggio di risposta alla sua richiesta di Join, cosi' che n venga a conoscenza di questo e quindi aggiorni $C(n)$ a $\{n, c\}$. Ad esempio, in Figura 4.5, il nodo n entra in contatto con il nodo c, che è il più vicino ad esso stesso.

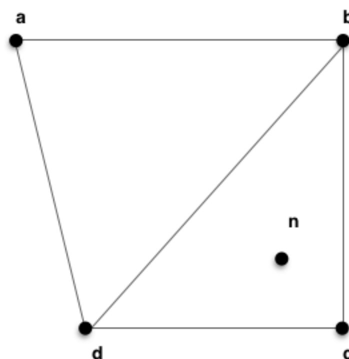

Figura 4.5: Posizionamento di n nella rete

A questo punto, il nodo n inizia lo scambio di messaggi con c per riempire l'insieme $C(n)$ e aggiorna $N^{queried}(n)$ a $\{c\}$. Il nodo c, ricevuta la richiesta del nodo n, se $n \notin C(c)$, lo aggiunge all'insieme $C(c)$ e aggiorna i propri vicini locali di Delaunay costruendo una triangolazione di Delaunay su $C(c)$ aggiornato. Quindi, calcola i vicini in comune con il nodo n e li invia in risposta alla richiesta di tale nodo.

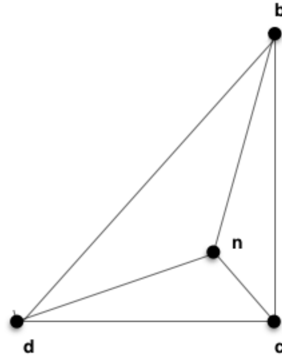

Figura 4.6: Interazione tra n e c

Come mostrato in Figura 4.6, il nodo c costruisce $DT(C(c))$ e calcola l'insieme di vicini della triangolazione di Delaunay in comune con n $N_{c,n}^{mutual}(c) = \{d, b\}$. Il nodo n usa l'informazione nella risposta del nodo c per aggiornare il proprio insieme $C(n)$ con tutti i nodi presenti nella risposta e per aggiornare i propri vicini di Delaunay con i nodi vicini di n nella triangolazione di Delaunay su $C(n)$. Per ridurre il numero di messaggi nella rete, ACE utilizza un'ottimizzazione basata sul fatto che *contattare un solo nodo per ogni nuovo triangolo è sufficiente alla convergenza dell'algoritmo*. Per questo motivo, i nuovi triangoli identificati in $DT(C(n))$ aventi n per vertice sono distinguibili in:

- triangoli controllati, i quali contengono almeno un nodo a cui n ha già inviato una richiesta;
- triangoli non controllati, i quali non contengono alcun nodo in $N^{queried}(n)$.

Ad esempio, in Figura 4.7, i triangoli Δbnc e Δdnc sono considerati controllati, perché n ha già contattato c, mentre il triangolo Δdnb non lo è.

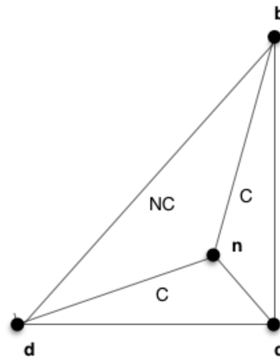

Figura 4.7: Individuazione triangoli controllati e non

Per ogni triangolo non controllato, viene scelto un vertice rappresentante a cui sarà inviata una nuova richiesta per nuovi vicini. A tutti gli altri nuovi vicini del nodo n viene inviato un messaggio per notificare della presenza di n.
Ad esempio, in Figura 4.8, n invia un messaggio a b per richiedere nuovi vicini, che considera la propria triangolazione di Delaunay e calcola i vicini comuni con n, $N_{b,n}^{mutual}(b) = \{c, a\}$.

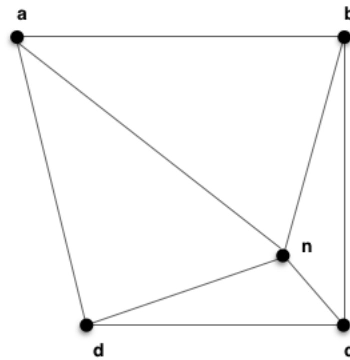

Figura 4.8: Richiesta di nuovi vicini

Quando tutti i nodi sono stati notificati e tutti i triangoli sono stati controllati, il nodo n ha terminato la fase di join nella rete.

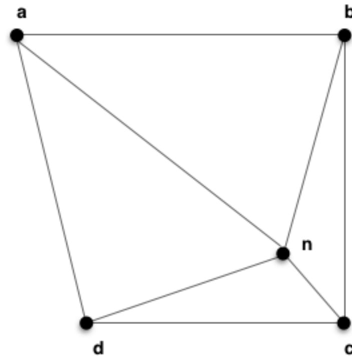

Figura 4.9: Triangolazione $DT(C(n))$ al termine della fase di Join

Per la fase di disconnessione volontaria, n costruisce la triangolazione di Delaunay sull'insieme $N(n)$, il quale contiene solamente i suoi vicini di Delaunay, e invia ad ogni nodo $v \in N(n)$ l'insieme di vicini di v in $DT(N(n))$, $N_v(n)$. Ogni nodo v utilizzerò l'informazione contenuta nel messaggio per aggiornare correttamente $C(v)$ eliminando il nodo n e aggiungendo eventuali nuovi nodi in $N_v(n)$, in modo da poter aggiornare i propri vicini di Delaunay.

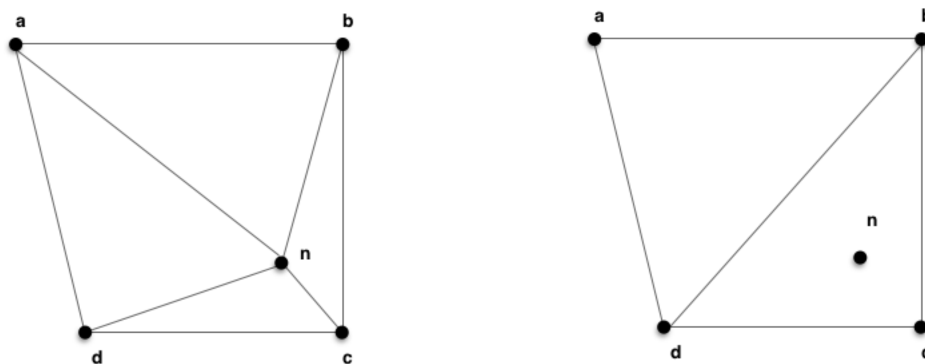

Figura 4.10: Disconnessione volontaria di n dalla rete

Come mostrato in Figura 4.10, il nodo n che sta per effettuare la disconnessione volontaria dalla rete calcola i seguenti insiemi:

- $N_a(n) = \{b, d\}$;
- $N_b(n) = \{a, c, d\}$;

- $N_c(n) = \{b, d\}$;
- $N_d(n) = \{a, b, c\}$;

I nodi a, b, c e d rimuovono n dal proprio insieme Candidate Set e aggiornano questo effettuando un'unione insiemistica tra i nodi inviati da n citati sopra e il proprio insieme Candidate Set. Infine, aggiornano la triangolazione di Delaunay locale, come mostrato dalla Figura 4.10 a destra.

Per la gestione dei fallimenti dei nodi, ACE prevede un meccanismo di Ping per ricercare eventuali nodi non più attivi nella rete. La gestione del fallimento di un nodo n è affidata a un suo vicino di Delaunay m detto *monitor node*. Il monitor node m memorizza gli insiemi $N_v(n)$, dove $v \in N(n)$, in modo da poter informare tutti i vicini di n in caso di un suo fallimento con tutte le informazioni necessarie per il corretto aggiornamento della loro triangolazione locale. Si noti che gli insiemi memorizzati da m sono gli stessi insiemi che calcolerebbe n nel caso di una disconnessione volontaria.
Il nodo m e la trasmissione degli insiemi citati a questo da parte di n sono effettuati ogni volta che l'insieme di vicini $N(n)$ viene aggiornato. Il nodo m, periodicamente, ha il dovere di verificare tramite messaggi di Ping se il nodo n di cui è responsabile è ancora partecipe nella rete. Allo scadere di un timeout apposito per n, se nessuna risposta da questo è pervenuta m invia a tutti i vicini di n gli insiemi che gli sono stati consegnati. Infine, aggiorna la propria triangolazione rimuovendo n da $C(m)$ e utilizzando l'insieme $N_m(n)$. Alla ricezione dell'insieme $N_v(n)$ in seguito a un fallimento di n, ognuno di questi aggiorna $C(v)$ rimuovendo n e aggiungendo eventuali nodi non conosciuti presenti in $N_v(n)$.

ACE prevede un meccanismo di manutenzione della rete al fine di riparare inconsistenze dovute al verificarsi di eventi concorrenti (Join, Leave, Failure). Il meccanismo è temporizzato e ha lo scopo per i nodi della rete di trovare nuovi potenziali vicini di cui questi non sono venuti a conoscenza. Ogni nodo n, allo scadere di un timer periodico, svuota il suo insieme di vicini interrogati $N^{queried}(n)$ e invia a ognuno dei nodi in $N(n)$ una nuova richiesta di vicinato.

Per la realizzazione della parte di comunicazione dei sotto-protocolli di ACE sono utilizzati i seguenti messaggi:

- il messaggio $Join$, inviato da un nuovo nodo n che intende entrare nella rete
- il messaggio di risposta alla $Join$, inviato dal nodo di *bootstrap*, il quale conterrà il nodo v più vicino al nodo n che vuole entrare nella rete, oppure un valore nullo nel caso in cui il nodo sia il primo della rete.
- il messaggio $NeighborSetRequest$, inviato dal nodo n che intende effettuare una richiesta di nuovi vicini. Questo messaggio viene inviato ai nodi v di cui n è venuto a

conoscenza che sono vertici di triangoli di cui anche il nodo n è vertice ma in cui il terzo vertice non è un nodo incluso nell'insieme $N^{queried}(n)$ in $DT(C(n))$. Lo scopo di tale messaggio è la conoscenza di nuovi nodi potenziali vicini, da aggiungere alla Candidate Set.

- il messaggio $NeighborSetReply$, inviato da n come risposta alla $NeighborSetRequest$ inviata da v e contenente l'insieme dei nodi vicini comuni a n e v in $DT(C(n))$.

- il messaggio $NeighborNotification$, inviato da un nodo n per notificare la sua presenza a tutti quei nodi v di cui n è venuto a conoscenza ma che stanno nello stesso triangolo di n e nodi in $N^{queried}(n)$ nella triangolazione di Delaunay $DT(C(n))$. In pratica, viene inviato a tutti quei nuovi nodi a cui non viene inviata una $NeighborSetRequest$.

- il messaggio *Leave*, inviato da un nodo n, il quale ha iniziato la fase di disconnessione volontaria dalla rete, a tutti i suoi vicini e contenente, per ogni vicino v, la lista di vicini di v in $DT(N(n))$, $N_v(n) = \{e | e \textit{ è un vicino di } v \textit{ in } DT(N(n))\}$. I nodi v utilizzano $N_v(n)$ per aggiornare $C(v)$ con nuovi potenziali vicini e di conseguenza per aggiornare la propria triangolazione di Delaunay locale. Inoltre, propagano la conoscenza dell'uscita di n utilizzando un protocollo di *Greedy Reverse Broadcast Path GRBP*.

- il messaggio *Delete*, inviato dai nodi v vicini di un nodo n che ha lasciato la rete per propagare l'informazione della sua uscita dalla rete.

4.2.1 Il protocollo ACE join

In questo paragrafo sarà presentato in dettaglio il protocollo di join descritto in [21]. Tra i messaggi presentati, il protocollo di Join utilizza i messaggi $NeighborSetRequest$, $NeighborSetReply$ e $NeighborNotification$.

Algorithm 7 *Join*(z) di un nodo n

1: **if** $z \neq NULL$ **then**
2: $\quad Send(z, NeighborSetRequest)$
3: $\quad C(n) = \{n, z\}; N^{queried}(n) = \{z\}; N(n) = \emptyset$
4: **else**
5: $\quad C(n) = \{n\}; N^{queried}(n) = \emptyset; N(n) = \emptyset$

L'algoritmo 7 mostra il funzionamento della fase di join eseguita da un nodo n. Il parametro z della funzione $Join$ rappresenta il nodo vicino più vicino restituito dal nodo di *bootstrap*. Se n è il primo nodo della rete, $z = NULL$.

L'algoritmo 8 mostra le azioni compiute dal nodo n alla ricezione del messaggio $NeighborSetRequest$ da parte di v: n controlla se v è contenuto nella sua Candidate Set $C(n)$ e nel caso non ci sia, lo aggiunge e aggiorna la propria triangolazione di Delaunay locale (calcolata in $C(n)$). In

Algorithm 8 Azioni compiute dal nodo n alla ricezione del messaggio $NeighborSetRequest(v)$

1: **if** $C(n)$ *not contains* v **then**
2: $C(n) = C(n) \cup \{v\}$
3: $N(n) =$ *neighbors of n on DT(C(n))*
4: $N^{mutual}_{n,v}(n) = \{e | e,\ v,\ n \text{ are in the same triangle in } DT(C(n))\}$
5: $Send(v, NeighborSetReply(N^{mutual}_{n,v}(n)))$

seguito, controlla i vicini comuni tra n e v e li aggiunge alla lista di vicini comuni $N^{mutual}_{n,v}(n)$ che sarà inviata a v tramite il messaggio $NeighborSetReply$. In questo modo il nodo v viene a conoscenza di nuovi vicini.

Algorithm 9 Azioni compiute dal nodo v alla ricezione del messaggio $NeighborSetReply(N^{mutual}_{n,v}(n))$ da parte di n

1: $C(v) = C(v) \cup N^{mutual}_{n,v}(n)$
2: $Update_neighbors(C(v), N(v))$

Algorithm 10 Procedura $Update_neighbors(C(v), N(v))$

1: $N^{old}(v) = N(v)$
2: $N(v) = neighbors\ of\ v\ on\ DT(C(v))$
3: $N^{new}(v) = N(v) - N^{old}(v)$
4: $T^{new}(v) = set\ of\ triangles\ that\ include\ v\ on\ DT(C(v))\ and\ don't\ include\ any\ node\ of\ N^{queried})(v)$
5: $N^{check}(v) = Get_neighbors_to_check(T^{new}(v))$
6: **for** *all neighbors n in* $N^{check}(v)$ **do**
7: $Send(n, NeighborSetRequest)$
8: $N^{querried}(v) = N^{querried}(v) \cup N^{check}(v)$
9: $N^{notify}(v) = N^{new}(v) - N^{check}(v)$
10: **for** *all neighbors n in* $N^{notify}(v)$ **do**
11: $Send(n, NeighborNotification)$

Gli algoritmi 9 e 10 mostrano le azioni compiute dal nodo v alla ricezione del messaggio $NeighborSetReply(N^{mutual}_{n,v}(n))$ da parte di n. Essenzialmente, il nodo v aggiorna la propria Candidate Set con i nuovi nodi ricevuti da n inclusi nel messaggio e ricalcola la sua triangolazione di Delaunay locale (su $C(v)$) dividendo i nuovi vicini in due liste:

1. la lista di nodi *checked* $N^{check}(v)$, contenente i nodi rappresentativi dei triangoli che non sono ancora stati controllati dal nodo v, cioè i triangoli, di cui v fa parte, che non contengono nodi contattati da v. Tali nodi sono scelti in modo casuale all'interno del triangolo non ancora controllato.

2. la lista di nodi *notify* $N^{notify}(v)$, contenente i nuovi nodi che non sono checked.

Ai nodi checked viene inviato un messaggio di $NeighborSetRequest$, mentre ai nodi notify un messaggio $NeighborNotification$.

Algorithm 11 Azioni compiute dal nodo n alla ricezione del messaggio $NeighborNotification(v)$

1: **if** $C(n)$ *not contains* v **then**
2: $\quad C(n) = C(n) \cup \{v\}$
3: $\quad N(n) =$ *neighbors of* n *on DT(*$C(n)$*)*

L'Algoritmo 11 descrive le azioni compiute dal nodo n alla ricezione del messaggio di $NeighborNotification$. n controlla se conosce già il nodo v: se lo conosce, non fa nulla, altrimenti lo aggiunge alla sua Candidate Set $C(n)$ e aggiorna la propria triangolazione di Delaunay locale.

4.2.2 Correttezza della procedura di Join sequenziale

Lee et al. dimostrano in [21] che il protocollo di join è corretto se si considerano join sequenziali, ovvero se durante l'esecuzione di una join non si verificano altre richieste di join, leave e failure. Per questo motivo, la suite di protocolli ACE è incompleta senza un protocollo di manutenzione, poiché non sarebbe applicabile all'interno di reti distribuite, quali P2P o reti di sensori, dove le operazioni avvengono in modo concorrente. Di seguito saranno enunciati i Teoremi e i Lemmi presentati in [21] relativi alla fase di join. I teoremi vengono riportati perché durante il lavoro di tesi ci si è resi conto che esiste uno scenario in cui un punto della dimostrazione del Lemma 4.2.2 non è verificato, come è mostrato nella sezione 4.3 relativa alle osservazioni sulla correttezza di ACE.

Lemma 4.2.1. *Sia n un nuovo nodo che sta eseguendo la procedura di join, S l'insieme di nodi esistenti nella rete, e $S' = S \cup \{n\}$. Supponiamo che la triangolazione di Delaunay distribuita $DT(S)$ sia corretta, e che nessun altro nodo effettui join, leave o failure. Sia T un triangolo che contiene n in $DT(C(n))$ ad un tempo t della procedura di join di n e che non esista in $DT(S')$. Sia $x \neq n$ un nodo in T. Supponiamo che n invii ad x una $NeighborSetRequest$. Dopo che n avrà ricevuto la $NeighborSetReply$ da x, T sarà rimosso da $DT(C(n))$.*

Lemma 4.2.2. *Sia n un nuovo nodo che sta per effettuare la join, S un insieme di nodi esistenti e $S' = S \cup \{n\}$. Supponiamo che la triangolazione di Delaunay distribuita su S, $DT(S)$, sia corretta, e che nessun'altro nodo effettui una join, leave o failure. Allora, il protocollo di join ACE termina e $C(n)$ conterrà tutti i vicini di n in $DT(S')$*

Dimostrazione. La seguente dimostrazione è presentata da Lee et al. in [21]. Essa segue i seguenti punti:

1. Si consideri un vicino v di n in $DT(S')$. Faremo vedere che v sarà incluso nella Candidate Set di n alla fine della procedura di Join.

2. Al passo 4 dell'algoritmo, n conterrà qualche nodo in $C(n)$ e calcolerà $DT(C(n))$.

3. Si suppone che a questo punto dell'esecuzione, il vicino v non sia ancora incluso in $C(n)$. Consideriamo la linea retta l da n a v.

4. Sia T il primo triangolo in $DT(C(n))$ che viene intersecato da l. Tale triangolo esiste perché v non è ancora un vicino di n in $DT(C(n))$. Inoltre, T è un triangolo che include n.

5. Siano gli altri nodi di T $x_1, x_2, \ldots, x_d$.

6. Per il Lemma precedente, l'esistenza di T implica che n non ha ancora ricevuto nessun messaggio $NeighborSetReply$ dai nodi in T.

7. T può includere un nodo x_i, $1 \leq i \leq d$ in $N^{queried}(n)$ oppure T non include nessun nodo in $N^{queried}(n)$. Nel primo caso, n ha inviato una $NeighborSetRequest$ a x_i, e riceverà una $NeighborSetReply$ da x_i. Nel secondo caso, al passo 5 dell'esecuzione del protocollo di join, n invierà una $NeighborSetRequest$ a un nodo x_j in T e riceverà da questo una $NeighborSetReply$. In entrambi i casi, quindi, per il Lemma precedente, T sarà rimosso da $DT(C(n))$.

8. Inoltre, se v continua a non essere un vicino di n in $DT(C(n))$, significa che l interseca un altro triangolo in $DT(C(n))$, quindi il protocollo continua la sua esecuzione. (Esegue il loop).

9. Questa procedura termina in un numero finito di iterazioni poiché il numero di nodi in S è finito e quindi lo è anche il numero di triangoli.

10. Quando non ci sono più triangoli intersecati da l in $DT(C(n))$, l è un arco di $DT(C(n))$ quindi v è un vicino di n e quindi è incluso in $C(n)$

□

Infine, il Teorema di correttezza per le join singole è il seguente:

Teorema 4.2.3. *(Correttezza protocollo ACE Join) Sia n un nuovo nodo che sta eseguendo la procedura di join, S un insieme di nodi già appartenenti alla rete e $S' = S \cup \{n\}$. Si supponga che nessun altro nodo esegua join, leave o failure, e che n esegua l'ACE join protocol. Allora, ACE join protocol termina e la triangolazione di Delaunay aggiornata è corretta.*

Dimostrazione. Vedi [21] per approfondimenti. Essenzialmente, la dimostrazione del teorema segue dai Lemmi presentati e dal fatto che se un nodo n conosce tutti i suoi vicini della triangolazione di Delaunay $DT(S)$ allora la sua triangolazione di Delaunay locale è corretta. □

4.2.3 Il protocollo ACE leave

La suite di protocolli ACE include un protocollo per la Leave volontaria dei nodi nella rete, in modo da mantenere la rete stabile anche durante l'uscita dei nodi. Così come il protocollo di join, il protocollo ACE leave è provato essere corretto per una singola leave, ma a differenza del primo, rimandiamo a [21] per i teoremi e le dimostrazioni.

Quando un nodo n vuole lasciare la rete, deve informare tutti i suoi vicini dell'uscita, inviando loro possibili nuovi nodi che potrebbero divenire loro vicini di Delaunay all'uscita di n.

Algorithm 12 *Leave*() di un nodo n

1: *Compute* $DT(N(n))$ *on* $N(n)$
2: **for** *all neighbor* v *on* $N(n)$ **do**
3: $\quad N_v(n) = \{w|w \text{ is a neighbor of } v \text{ on } DT(N(n))\}$
4: $\quad Send(v, Leave(N_v(n)))$

L'Algoritmo 12 mostra le azioni compiute dal nodo n che sta per effettuare la Leave. Il nodo n si limita a calcolare la triangolazione di Delaunay dei suoi vicini senza se stesso e inviare i vicini dei propri vicini a questi. Questo viene effettuato perché l'uscita di n può far si che qualche nodo non conosciuto precedentemente dai vicini v di n sia diventato un nuovo vicino di v.
Alla ricezione del messaggio, il vicino v di n aggiorna la propria Candidate Set eliminando n da questa e aggiungendo eventuali nuovi nodi presenti nella lista contenuta nel messaggio e aggiorna la propria triangolazione locale di Delaunay. In seguito (Algoritmo 13) v propaga l'uscita di n dalla rete inviando nella rete dei messaggi *Delete* allo scopo di informare i nodi che hanno n nella propria *CandidateSet* in modo che rimuovono n dalla propria Candidate Set.

Algorithm 13 v riceve un messaggio $Leave(N_v(n))$ da un nodo n

1: $C(v) = (C(v) \cup N_v(n)) - \{n\}$
2: $N(v) = \text{neighbors of } v \text{ on } DT(C(v))$
3: $GRBP(Delete(n), n)$

Come si vede dall'Algoritmo 14, la differenza tra il messaggio *Delete* e quello *Leave* è che non viene aggiornata la lista dei vicini dei nodi che ricevono il messaggio *Delete*.

Algorithm 14 w riceve un messaggio $Delete(n)$ da un nodo v

1: $C(w) = C(w) - \{n\}$
2: $GRBP(Delete(n), n)$

Algorithm 15 $GRBP(m, s)$ eseguito da u

1: $C(w) = C(w) - \{n\}$
2: $GRBP(Delete(n), n)$

Infine, l'algoritmo 15 descrive il funzionamento della funzione Greedy Reverse Broadcasting Path, dove i parametri m e s rappresentano rispettivamente il messaggio da propagare nella rete (nel nostro caso un messaggio *Delete*) e la sorgente del messaggio, il quale è il primo nodo che ha generato il messaggio *Delete*.

4.2.4 Il protocollo ACE failure

Oltre ai protocolli di join e di leave, la suite ACE fornisce un protocollo per la gestione e il rilevamento di fallimenti di nodi all'interno della rete.

Tale protocollo è stato introdotto al fine di ridurre il numero di esecuzioni del protocollo di Manutenzione, anche se quest'ultimo continua ad essere necessario per la gestione di eventi concorrenti.

Il protocollo ACE failure funziona tramite la definizione di un *Piano di Contingenza*, un piano da eseguire per ogni nodo in caso di fallimento, e la sua consegna a un nodo speciale (*monitor node*) ogni volta che avviene qualche modifica nella lista dei vicini del nodo proprietario del Piano di Contingenza.
Il monitor node m di un nodo v è un vicino speciale che avrà il compito di controllare che v sia ancora attivo nella rete a determinati intervalli di tempo, di mantenere in memoria il piano di contingenza per v e di eseguirlo nel caso in cui v non sia più attivo. Tale nodo può essere scelto casualmente, ma nell'algoritmo è specificato essere il vicino con l'identificatore più piccolo (applicabile in caso di identificatore numerico o stringa).

Nel piano di contingenza viene inserita una mappa $< n, N_v(n) >$, la quale memorizza l'insieme di vicini dei vicini n del nodo v in $DT(N(v))$. Si noti che tale mappa rappresenta le liste di vicini che venivano inviate con i messaggi di *Leave* spiegate nel paragrafo precedente.

Per controllare che il nodo v sia ancora attivo, allo scadere di timeout stabiliti, il nodo monitor m invia a v un messaggio di *Ping*, il quale sarà seguito da un messaggio di *Pong* inviato da v in caso sia ancora attivo. Nel caso in cui v non sia attivo, m provvede a inviare dei messaggi $Failure(N_v(n))$ ai nodi n vicini di v, utilizzando le informazioni del piano di

contingenza.

Infine, m rimuove v dalla propria Candidate Set, aggiorna i propri vicini e propaga il messaggio *Delete* nella rete.

In sintesi, quello che viene effettuato dal protocollo di Failure è esattamente quello che avviene tramite il protocollo di Leave, solo che viene gestito da un nodo m diverso dal nodo v che effettua l'uscita dalla rete, poiché tale nodo è impossibilitato a causa del fallimento.

Algorithm 16 Ogni volta che cambia N_u

1: m_u = *neighbor node with least ID on* $N(u)$
2: *compute* $DT(N(u))$
3: **for** *all neighbors* v *on* $N(u)$ **do**
4: $N_v(u)$ = *neighbor nodes of* v *on* $DT(N(u))$
5: $Send(m_u, Contingency_Plan(\{< v, N_v(u) > | v \in N(u)\}))$

Algorithm 17 Ricezione del $Contingency_Plan(CP_u)$ da u in m

1: *Set the timeout timer for* u *to* $T + F$

Algorithm 18 Scadenza del timeout timer per u in m

1: $Send(u, Ping)$
2: *Set* $Ping_timeout_timer$ *to* $T + TO$

Il protocollo di Failure di ACE non è stato implementato nel codice finale. Questa scelta è motivata dal fatto che l'algoritmo col quale si vuole fare il confronto finale non distingue la Leave dalla Failure, quindi si è preferita l'implementazione di una Leave volontaria perché più semplice da realizzare e da simulare.

4.2.5 Il protocollo di Manutenzione ACE

L'ultimo tassello della suite di protocolli ACE è il *Protocollo di Manutenzione*, il quale risulta essere necessario (vedi [21]) per il corretto funzionamento del sistema con inserimenti, leave e failure concorrenti.
Esso ha due scopi:

1. permettere la conoscenza di nuovi vicini ai nodi, poiché se tutti i nodi contengono tutti i suoi vicini di Delaunay nella Candidate Set è dimostrato essere corretta la triangolazione di Delaunay distribuita ([21]);

Algorithm 19 Ricezione del $Pong(flag)$ da u in m

1: *cancel* $Ping_timeout_timer$
2: **if** $flag = true$ **then**
3: *Set the timeout timer for u to* $T + F$
4: **else**
5: *cancel timeout timer*

Algorithm 20 Scandenza del Ping timeout timer per u in m

1: *cancel* $Ping_timeout_timer$
2: **for** *all nodes w in* CP_u $(< w, N_w(u) >)$ **do**
3: $Send(w, Failure(N_w(u)))$
4: $C(m) = (C(m) \cup N_m(u)) - \{u\}$
5: $N(m) =$ *neighbors node of m on* $DT(C(m))$
6: $GBRP(u, Delete(u))$

2. correggere la rete nel caso in cui il protocollo di failure si sia perso qualche fallimento di qualche nodo. Tale scopo risulta essere non fondamentale in quanto il protocollo di Failure è già sufficiente per la gestione di questi casi.

Nella implementazione effettuata, non avendo implementato il protocollo di Failure, è il protocollo di Manutenzione a gestire i casi dei nodi falliti o andati via in modo concorrente con altri nodi.

Siccome lo scopo principale del protocollo è quello di conoscere nuovi nodi, questo semplicemente azzera, allo scadere di un timer periodico, la lista dei nodi interrogati di ogni nodo e procede con l'invio di nuove $NeighborSetRequest$ ai nodi che sa essere suoi vicini, scegliendo accuratamente un nodo per ogni triangolo (ottimizzazione al fine di limitare il numero di messaggi nella rete).

Per lo scopo secondario, invece, all'invio della $NeighborSetRequest$ verso un nodo v ogni nodo setta un timer $Timeout_v$: se viene ricevuta risposta da tale nodo, allora non viene fatto nulla, altrimenti significa che il nodo v ha subito un fallimento, quindi viene inviato un messaggio *Remove* nella rete per comunicare ai nodi che il nodo v è da rimuovere perché non è più attivo nella rete.

Algorithm 21 Recezione del messaggio $Failure(u, N_u(v))$ da m

1: $C(v) = (C(v) \cup N_v(u)) - \{u\}$
2: $N(v) =$ *neighbors node of v on* $DT(C(v))$
3: $GBRP(u, Delete(u))$

Algorithm 22 Esecuzione del protocollo di Manutenzione da parte del nodo n

1: $N^{querried}(n) = \emptyset$
2: $T = set\ of\ triangles\ that\ contains\ n\ on\ DT(N(n) \cup \{n\})$
3: $N^{check}(n) = Get_neighbors_to_check(T)$
4: **for** *each neighbor* v *in* $N^{check}(n)$ **do**
5: $Send(v, NeighborSetRequest(n))$
6: *set* $Timeout_v$ *to* $CURRENT + DELTA_v$
7: *set* $Timeout$ *to* $CURRENT + PERIOD_TIMER$

4.3 Osservazioni sulla correttezza di ACE

Analizzando l'algoritmo e con l'aiuto dell'implementazione è stata verificata l'esistenza di alcuni casi in cui l'algoritmo ACE non è in grado di garantire la correttezza della triangolazione di Delaunay distribuita in uno spazio a due dimensioni considerando join sequenziali.
In particolare, è possibile che alcuni nodi che effettuano la join in una rete con una triangolazione corretta al tempo t non vengano a conoscenza di tutti i loro vicini, quindi esistono alcuni casi in cui il teorema di correttezza 4.2.3 non sia corretto. In particolare si consideri il seguente esempio:

Esempio. Si consideri la seguente triangolazione di Delaunay al tempo t (descritta in Figura 4.11) calcolata su $S = \{1, 2, 3, 4\}$. Si supponga che al tempo t la triangolazione di Delaunay sia corretta, e che quindi valgano le seguenti condizioni:

- $N(1) = \{2, 3\}$; $\{1, 2, 3\} \subset C(1)$; $\{1, 2, 3\} \subset N(1)^{queried}$;
- $N(2) = \{1, 3, 4\}$; $C(2) = S$; $N(2)^{queried} = S$;
- $N(3) = \{1, 2, 4\}$; $C(3) = S$; $N(3)^{queried} = S$;
- $N(4) = \{2, 3\}$; $\{2, 3, 4\} \subset C(4)$; $\{2, 3, 4\} \subset N^{queried}(4)$;

Ora supponiamo che al tempo $t + 1$ un nuovo nodo 5 intenda entrare nella rete e posizionarsi come mostrato in Figura 4.12.

Appena entrato nella rete, il nodo 5 contatta il nodo di *bootstrap* per venire a conoscenza del nodo più vicino a lui nella rete, il quale è il nodo 3. Quindi, 5 invia a 3 una $NeighborSetRequest$ e aggiorna la propria Candidate Set e la Neighbor Querried Set ($C(5) = \{3, 5\}, N^{queried}(5) = \{3\}$).
Il nodo 3, che riceve il messaggio, aggiunge 5 alla Candidate Set ($C(3) = \{1, 2, 3, 4, 5\}$) e calcola i suoi nuovi vicini nella triangolazione di Delaunay $DT(C(3))$, i quali risultano essere come mostrato in Figura 4.13 $N(3) = \{2, 4, 5\}$.

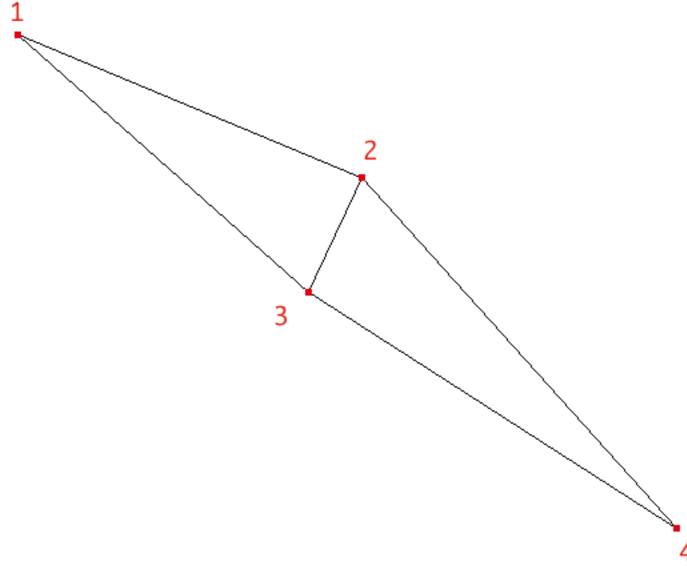

Figura 4.11: La triangolazione di Delaunay su S al tempo t

.

Il nodo 3, quindi, calcola i vicini comuni tra 3 e 5 in $DT(C(3))$ e li invia a 5 tramite il messaggio $NeighborSetReply$. Si noti che la triangolazione $DT(C(3))$ è corretta e coincide con la triangolazione di Delaunay su S, perché $C(3) = S$. I nodi che vengono inclusi nel messaggio $NeighborSetReply$ sono $N(5)^{mutual}_{3,5}(3) = \{2, 4\}$. *Si noti anche che a causa del flip dell'arco che collegava* 3 *e* 1*, il nodo* 1 *non viene notificato a* 5.

Quando 5 riceve il messaggio $NeighborSetReply$, aggiunge i nuovi nodi 2 e 4 alla sua Candidate Set ($C(5) = \{2, 3, 4, 5\}$) e aggiorna la propria triangolazione di Delaunay locale ($DT(C(5))$), come mostrato in Figura 4.14.

Quindi, il nodo 5 esegue la procedura $Update_neighbors$: si ricorda che in tale procedura vengono distinti i triangoli che non contengono alcun nodo in $N^{queried}(5)$ dagli altri in $DT(C(5))$, e da questi triangoli viene preso un nuovo nodo per triangolo a cui inviare una $NeighborSetRequest$, mentre agli altri nuovi nodi sarà inviata una semplice notifica $NeighborNotification$. Poiché tutti e 3 i triangoli della triangolazione di Delaunay $DT(C(5))$ includono il nodo 3, il quale è in $N^{queried}(5)$, tutti i triangoli della triangolazione calcolata sono stati già selezionati, quindi nessuno dei nodi nuovi ($N^{new}(5) = \{2, 3, 4\}$) è un nodo *checked*.

A questo punto, le liste del nodo 5 sono aggiornate in questo modo:

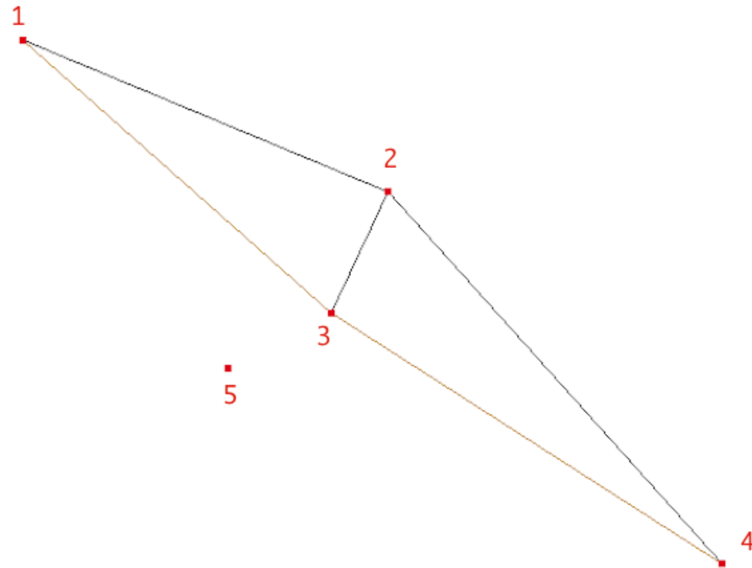

Figura 4.12: Il nuovo nodo 5 posizionato nella rete

.

- $C(5) = \{2, 3, 4, 5\}$
- $N(5) = \{2, 3, 4\}$
- $N^{queried}(5) = \{3\}$

Quindi, i nodi notify sono 2 e 4, mentre non esistono nodi in $N^{check}(5)$. 5 invia quindi a 2 e a 4 una $NeighborNotification$.

Nello step successivo, 2 e 4 ricevono una $NeighborNotification$ da parte di 5, quindi aggiungono 5 a $C(2)$ e $C(4)$ e aggiornano $N(2)$ e $N(4)$ calcolando la triangolazione sulle Candidate Set. L'algoritmo cosi termina, lasciando la triangolazione distribuita in uno stato errato, poiché ne il nodo 5 ne 1 sono venuti a conoscenza l'uno dell'altro. La Figura 4.3 mostra la triangolazione distribuita calcolata al termine della procedura di join del nodo 5: il numero di errori totale è 2. Questo problema si ripercuote anche sugli inserimenti successivi, che non trovando una triangolazione corretta non riescono a loro volta a costruire una triangolazione di Delaunay locale corretta.

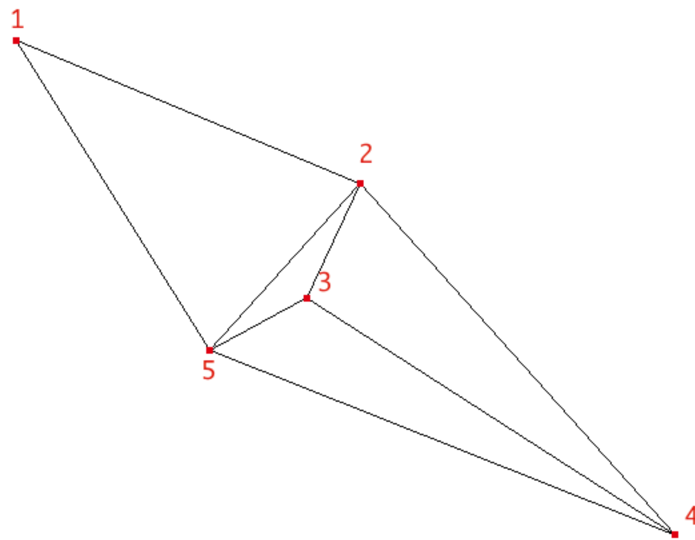

Figura 4.13: La triangolazione di Delaunay $DT(C(3))$ dopo l'aggiunta del nodo 5 a $C(3)$
.

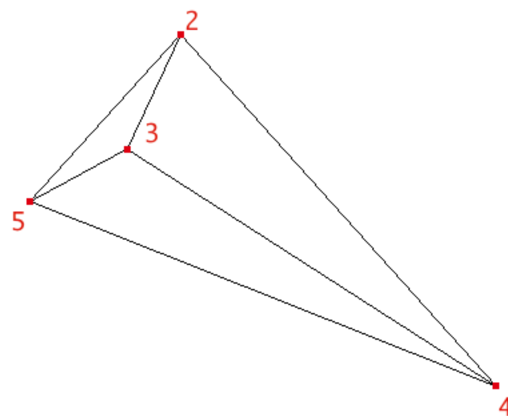

Figura 4.14: La triangolazione di Delaunay $DT(C(5))$ dopo la ricezione di $NeighborSetReply(N(5)_{3,5}^{mutual}(3))$
.

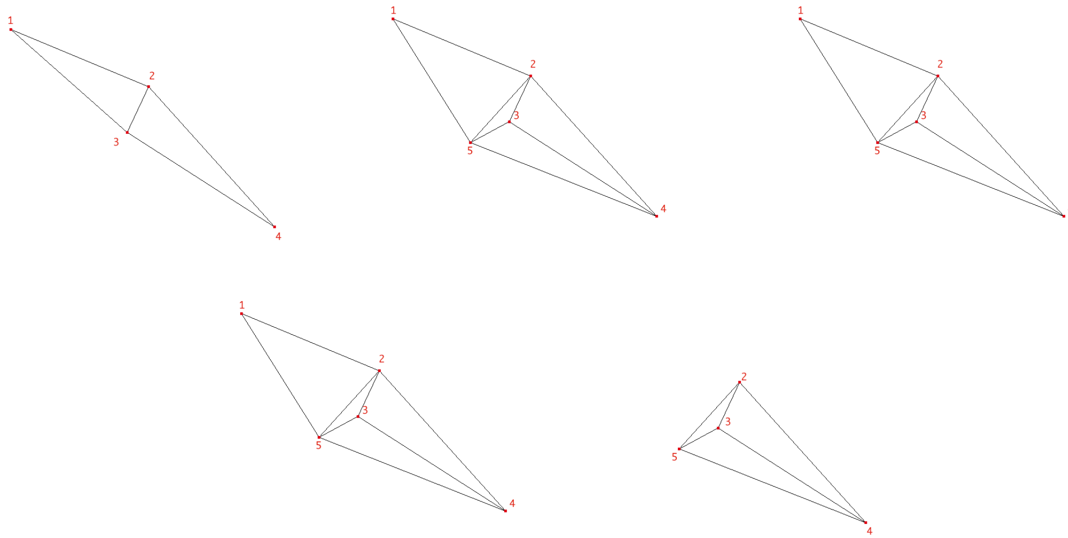

Figura 4.15: In ordine dall'alto verso il basso, da sinistra a destra: le triangolazioni di Delaunay al termine della procedura di join del nodo 5 calcolate in $C(1)$, $C(2)$, $C(3)$, $C(4)$, $C(5)$

.

Analisi del problema. È stata quindi effettuata un analisi sul teorema di correttezza (Teorema 4.2.3) e dei Lemmi 4.2.1 e 4.2.2 al fine di verificare il corretto funzionamento della procedura di join sequenziale. Nell'analisi effettuata, si è trovata un'assunzione errata relativa al punto 4 della dimostrazione, il quale identifica un triangolo T intersecato dalla linea retta che collega il nuovo vicino non ancora aggiunto v con il nodo n che stiamo considerando.
Secondo Lee et al. in [21], tale triangolo esiste sempre: questo è verificato essere non sempre vero. Si consideri, ad esempio, l'esempio presentato precedentemente, il quale nonostante la sequenzialità della procedura di join del nodo 5 termina con 2 errori, poiché il nodo 5 e 1 non vengono a conoscenza l'uno dell'altro. Adattando la dimostrazione del Lemma 4.2.2, il nostro nodo n è il nodo 1, mentre il nodo v è il nostro 5: supponiamo inoltre che il nodo 5 abbia già inviato la $NeighborSetRequest$ a 3 e che abbia già ricevuto la $NeighborSetReply$ e abbia già calcolato la triangolazione di Delaunay in $C(5)$.

A questo punto, consideriamo la linea retta l che congiunge 5 a 1, così come mostrato in 4.16.

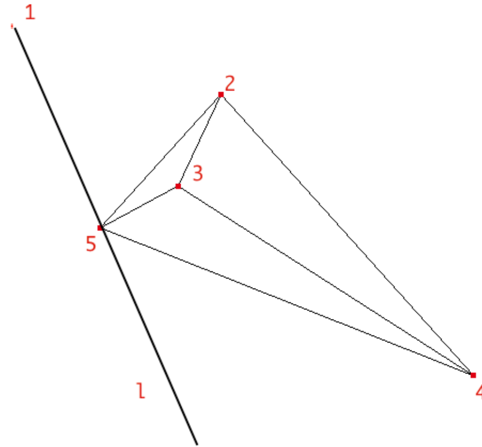

Figura 4.16: In figura: La linea retta tracciata da 1 a 5 non attraversa alcun triangolo nella triangolazione $DT(C(5))$

.

Secondo il punto 4 della dimostrazione del Lemma 4.2.2, l dovrebbe intersecare un triangolo nella triangolazione di Delaunay $DT(C(5))$, ma come si evince dalla Figura 4.16, questo non è vero in questo caso. Quindi, non è possibile applicare il Lemma 4.2.1, e di conseguenza l'algoritmo di join termina senza che il nodo 5 venga a conoscenza di tutti i suoi vicini di Delaunay, in particolare del nodo 1.

Elaborazione di una soluzione. A causa dei problemi individuati grazie all'analisi effettuata, è nata l'esigenza di elaborare un nuovo algoritmo che, a differenza di ACE, utilizza un numero maggiore di messaggi, ma raggiunge l'accuratezza del 100%. Tale algoritmo sarà descritto nel prossimo paragrafo.

4.4 La suite di protocolli New ACE

In questa Sezione sarà descritto *New ACE*, il nuovo protocollo derivante dalle modifiche apportate ad ACE. È necessario effettuare le seguenti considerazioni:

- la nuova suite di protocolli è *diversa* rispetto a ACE, nonostante ne condivida alcuni messaggi e alcuni protocolli: infatti, i protocolli di Leave e Failure rimangono invariati, mentre il protocollo di Join viene modificato con nuovi messaggi che sostituiscono $NeighborSetRequest$, $NeighborSetReply$ e $NeighborNotification$.
- la suite di protocolli è corretta per join, leave e failure non concorrenti. Il protocollo di Manutezione è ancora necessario per la correttezza con join, leave e failure concorrenti.

Il protocollo New ACE ha un funzionamento simile ad ACE, ma si differenzia per il protocollo di Join e il protocollo di Manutenzione. Essi prevedono l'utilizzo dei seguenti nuovi messaggi:

1. il messaggio $NeighborRequest$, utilizzato in sostituzione della $NeighborSetRequest$, con parametri:
   (a) u: il nodo che effettua la richiesta di vicinato
   (b) $N_{u,v}^{mutual}(u)$: i nodi che, secondo u, sono i vicini comuni tra u e v, dove v è il nodo a cui è inviata la richiesta di vicinato.
2. il messaggio $NeighborReply$, utilizzato in sostituzione della $NeighborSetReply$, con parametri:
   (a) u: il nodo che invia il messaggio.
   (b) $N_{u,v}^{mutual}(u)$: i nodi che, secondo u, sono i vicini comuni tra u e v, dove v è il nodo a cui è inviata la risposta.
3. il messaggio *Acknowledge*, utilizzato per terminare la conversazione tra due nodi u e v, con parametro:
   (a) u: il nodo che invia il messaggio.
4. il messaggio $NotNeighbor$, utilizzato da un nodo v come risposta al messaggio di $NeighborSetRequest$ inviato da u nel caso in cui v e u non siano vicini, con parametri:
   (a) v: il nodo che invia il messaggio
   (b) $PN_v(u)$: la lista dei nodi conosciuti da v, possibili vicini per u, definita come:

$$PN_v(u) = \{e \in C(v) | d(e, v) \leq d(u, v)\} \tag{4.1}$$

   dove la funzione $d(a, b)$ rappresenta la distanza euclidea tra due nodi a e b nella rete.

Il nuovo protocollo è molto più semplice del protocollo originale: quando un nodo n vuole entrare nella rete, contatta il nodo di *bootstrap* per conoscere il suo vicino più vicino v. In seguito, vengono eseguiti i seguenti passaggi (i passi 3-4 vengono eseguiti in loop finché vengono ricevute nuove $NeighborReply$):

1. il nodo n invia una $NeighborRequest(n, [])$ a v. Nota: la lista dei vicini comuni a v e n secondo n è vuota.

2. il nodo v riceve la $NeighborRequest(n, [])$: controlla quindi se n è presente nella sua Candidate Set $C(v)$ e, se non lo è, lo aggiunge e aggiorna i propri vicini di Delaunay costruendo la triangolazione di Delaunay locale su $C(v)$. Quindi, controlla se il nodo $n \in N(v)$:
   - se $n \in N(v)$, allora costruisce la lista $N_{v,n}^{mutual,new}(v)$ dei vicini comuni a n e v che n ancora non conosce e, se non è vuota, invia un messaggio di riposta $NeighborReply(v, N_{v,n}^{mutual,new}(v))$ a n, altrimenti notifica n con un messaggio $Acknowledge(v)$.
   - altrimenti, notifica n del fatto che non sono vicini con un messaggio $NotNeighbor(v, PN_n(v))$, dove $PN_n(v)$ rappresenta la lista dei nodi $e \in C(v)$ definita come $PN_n(v) = \{e \in C(v) | d(e, v) \leq d(u, v)\}$, dove $d(-, -)$ è la funzione distanza euclidea.
3. n riceve un messaggio da v: questo messaggio può essere:
   - un messaggio $NeighborReply(v, L)$ o un messaggio $NotNeighbor(v, L)$: in questo caso, n aggiorna la propria Candidate Set $C(n)$ aggiungendo i nodi in L, e aggiorna i propri vicini di Delaunay nella triangolazione $DT(C(n))$.
   - un messaggio $Acknowledge(v)$: in questo caso, n aggiorna la propria Candidate Set aggiungendo v se necessario, e aggiornando i propri vicini di Delaunay in $DT(C(n))$.
4. se ha ricevuto un messaggio $NeighborReply(v, L)$ o un messaggio $NotNeighbor(v, L)$ al passo 3, per ogni nuovo vicino w, n calcola la lista di nodi comuni con w, $N_{n,w}^{mutual}(n)$, e invia una $NeighborRequest(n, N_{n,w}^{mutual}(n))$ a w.

Algorithm 23 $Join(z)$ di un nodo n in New ACE

```
1: if z ≠ NULL then
2:     Send(z, NeighborRequest(z, []))
3:     C(n) = {n, z}; N(n) = ∅
4: else
5:     C(n) = {n}; N(n) = ∅
```

Gli Algoritmi 23, 24 e 25 mostrano le azioni compiute dai nodi nel momento in cui si vuole effettuare una join nella rete o quando vengono ricevuti i nuovi messaggi $NeighborRequest$ e $NeighborReply$. Questi algoritmi risultano essere efficaci perché ogni nodo viene a conoscenza di tutti i suoi vicini di Delaunay tramite lo scambio dei nuovi messaggi.

L'algoritmo 26 mostra il funzionamento del protocollo di manutenzione eseguito da un nodo n: a differenza del protocollo di manutenzione di ACE, in New ACE semplicemente

Algorithm 24 Azioni compiute dal nodo n alla ricezione del messaggio $NeighborRequest(v, N^{mutual}_{v,n}(v))$

1: **if** $C(n)$ *not contains* v **then**
2: $C(n) = C(n) \cup \{v\}$
3: $N(n) =$ *neighbors of* n *on* $DT(C(n))$
4: $N^{mutual}_{v,n}(n) = \{e | e,\ v,\ n$ *are in the same triangle in* $DT(C(n))\}$
5: $N^{mutual,new}_{v,n}(n) = N^{mutual}_{v,n}(n) - N^{mutual}_{v,n}(v))$
6: $Send(v, NeighborReply(n, N^{mutual,new}_{v,n}(n))$

Algorithm 25 Azioni compiute dal nodo v alla ricezione del messaggio $NeighborReply(n, N^{mutual,new}_{v,n}(n))$ da parte di n

1: $C(v) = C(v) \cup N^{mutual,new}_{v,n}(n)$
2: $N^{old}(v) = N(v)$;
3: $N(v) =$ *delaunay neighbors of* v *on* $DT(C(v))$
4: $N^{new}(v) = N(v) - N^{old}(v)$
5: **for** *each node* w *on* $N^{new}(v)$ **do**
6: $N^{mutual}_{v,w}(v) = \{e | e,\ v,\ w$ *are in the same triangle in* $DT(C(v))\}$
7: $Send(w, NeighborRequest(v, N^{mutual}_{v,w}(v)))$

ogni nodo n invia una nuova richiesta di vicinato a tutti i vicini conosciuti. In questo modo, grazie ai messaggi $NeighborReply$ e $NotNeighbor$, viene effettuato uno scambio di messaggi che permette la correzione della rete. In seguito, viene mostrata in un esempio l'esecuzione della join di New ACE.

Algorithm 26 Esecuzione del protocollo di Manutenzione da parte del nodo n per New ACE

1: **for** *each neighbor* v *in* $N(n)$ **do**
2: *set* $Timeout_v$ *to* $CURRENT + DELTA_v$
3: $Send(v, NeighborRequest(n, []))$
4: *set* $Timeout$ *to* $CURRENT + PERIOD_TIMER$

Esempio. Si consideri la configurazione della rete mostrata in Figura 4.17(a destra) e un nuovo nodo 5 entrato nella rete. Si supponga che il nodo 5 abbia individuato il nodo 3 come suol nodo più vicino e che gli abbia inviato un messaggio $NeighborRequest(5, [])$.

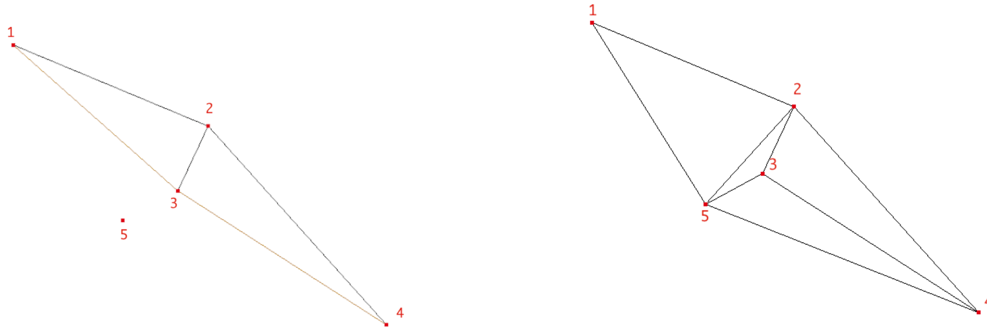

Figura 4.17: Posizionamento del nodo 5 nella rete e triangolazione di Delaunay locale del nodo 3 sopo l'aggiunta del nodo 5
.

Il nodo 3 riceve la $NeighborRequest$, aggiunge 5 alla sua Candidate Set ($C(3) = S \cup \{5\}$) e aggiorna i propri vicini di Delaunay, come mostrato in Figura 4.17(a destra). A questo punto, identifica i suoi vicini comuni con 5 che non siano già conosciuti da 5. I nuovi vicini quindi calcolati da 3 sono:

$$N_3^{new}(5) = N_3^5 - [] = \{2\} \cup \{4\} = \{2, 4\} \tag{4.2}$$

Quindi, 3 invia una $NeighborReply(3, \{2, 4\})$ a 5.
Il nodo 5 riceve la $NeighborReply$ da parte di 3 e aggiorna la propria Candidate Set, la quale diventa $C(5) = \{5, 2, 3, 4\}$. Quindi calcola la triangolazione di Delaunay e i nuovi vicini, i quali risultano essere $\{2, 3, 4\}$, come mostrato in Figura 4.18. Per ognuno di essi, calcola i vicini comuni e loro invia una nuova $NeighborRequest$. Nota: il nodo 3 ha già ricevuto tale messaggio, ma gli viene inviato nuovamente perché la prima volta non era stato aggiunto alla lista di vicini. Un'ottimizzazione potrebbe essere quella di ricordare i nodi contattati (utilizzando per esempio la $N^{queried}(5)$).

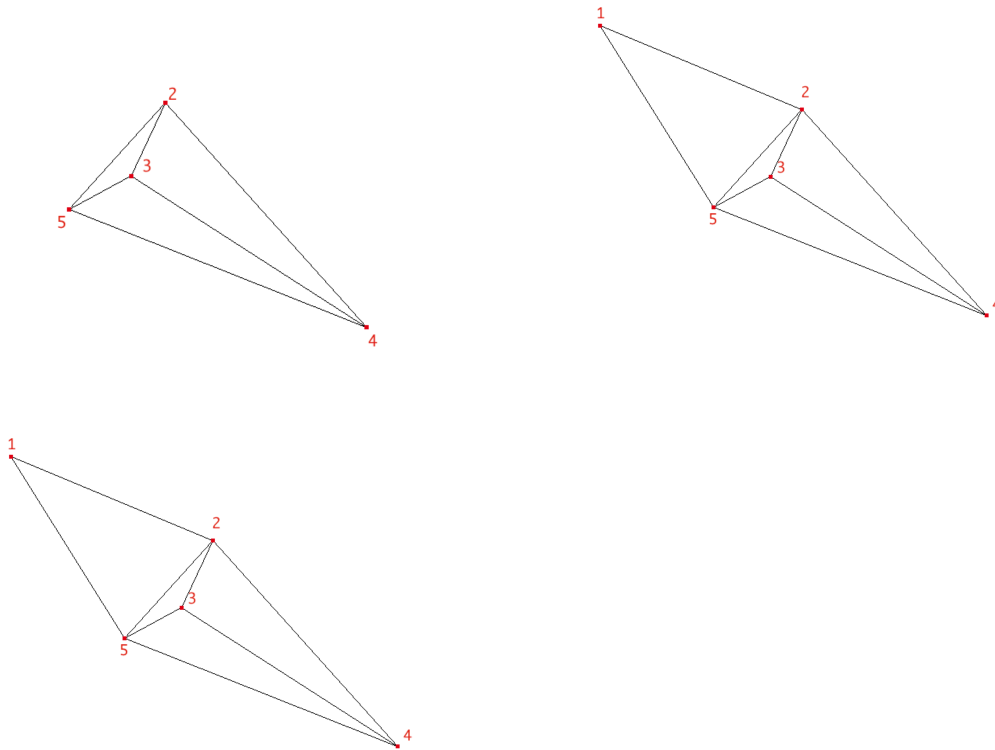

Figura 4.18: Triangolazioni locali dei nodi 2,4 e 5 nel prossimo passo di esecuzione dell'algoritmo
.

Quindi, i nodi 2 e 4 ricevono rispettivamente i messaggi $NeighborRequest(5, \{4, 3\})$ e $NeighborRequest(5, \{2, 3\})$, aggiungono 5 alla propria Candidate Set e calcolando $DT(C(2))$ e $DT(C(4))$(Figura 4.18). Le triangolazioni calcolate da 2 e 4 sono corrette: il nodo 4 calcola i vicini comuni con 5 che 5 non conosce e scopre che non ce ne sono, quindi non invia risposta a 5. Al contrario, 2 scopre che esiste il nodo 1 comune con 5 che 5 non conosce: quindi, invia una $NeighborReply(2, \{1\})$ a 5.
Quindi 5 riceve il messaggio $NeighborReply(2, \{1\})$ da 2, aggiunge 1 alla propria Candidate Set e calcola $DT(C(5))$ (Figura 4.19).

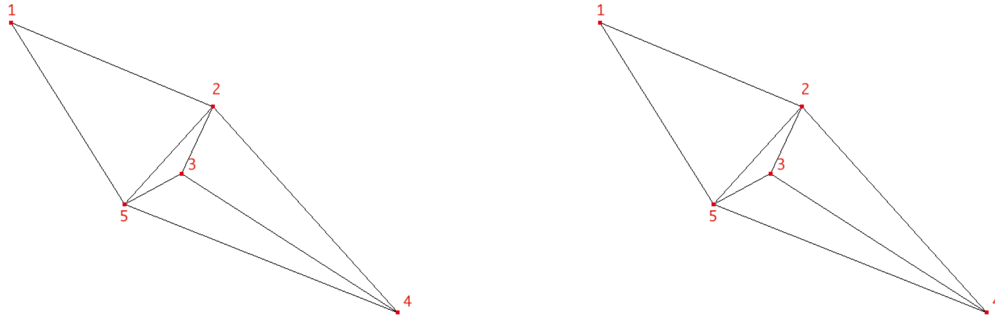

Figura 4.19: Triangolazione locale di 1 e 5 nella fase finale della join di 5

.

Considerando che l'unico nuovo vicino aggiunto da 5 è 1, 5 calcola i vicini comuni tra 5 e 1 ($\{2\}$) e invia una $NeighborRequest(5, \{2\})$ a 1.
Il nodo 1 riceve il messaggio, quindi viene a conoscenza di 5, aggiorna la propria Candidate Set e calcola $DT(C(1))$(4.19). Quindi, 1 calcola i vicini comuni tra 1 e 5 non noti a 5 in $DT(C(1))$, scoprendo che non ce ne sono (l'unico vicino comune è 2 ma 5 lo conosce già), quindi l'algoritmo termina, e la triangolazione di Delaunay distribuita è corretta.

Capitolo 5

New ACE: Implementazione

In questo capitolo sono presentate le classi e le strutture utilizzate per l'implementazione degli algoritmi ACE e New ACE al fine di effettuare i test descritti nel prossimo capitolo. L'implementazione è stata realizzata in Java utilizzando il simulatore di reti Peer to Peer *PeerSim* (peersim.sourceforge.net).

5.1 PeerSim

PeerSim [29] è un simulatore open source per sistemi P2P sviluppato dall'Università di Bologna scritto in Java, con il quale è possibile implementare protocolli P2P. PeerSim è altamente scalabile e single thread, quindi si presta bene al test di reti con molti nodi (fino a un numero di nodi dell'ordine di 10^5).
Componenti chiave di PeerSim sono :

- Network: si tratta di un array globale utilizzato per memorizzare i nodi che sono nella rete. Il numero dei nodi all'interno dell'array viene definito nel *file di configurazione*.
- Node: classe che modella i nodi della rete, memorizza i protocolli in una coda che viene utilizzata ad ogni ciclo per accedere ai protocolli ed effettuare le operazioni locali. La classe node contiene un metodo $getProtocol(intPid)$ il quale viene utilizzato per accedere ai protocolli del nodo.
- Interfaccia Linkable: viene utilizzata per la rappresentazione della vista del nodo, quindi memorizza liste di nodi conosciuti e vicini, e effettua operazioni su di esse.
- Interfaccia Transport: viene utilizzata per l'invio dei messaggi nella rete, utilizzando il metodo *send*().

- Protocolli: vengono utilizzati per la descrizione delle azioni che ogni nodo deve compiere ad ogni ciclo. Un ciclo è un intervallo di tempo predefinito descritto nel *file di configurazione*. Vengono utilizzati per descrivere le azioni locali dei nodi.
- Controlli: servono a compiere azioni di natura globale nella rete, per esempio analisi di performance o inizializzazione.
- File di Configurazione: si tratta di un file di testo utilizzato da PeerSim per il settaggio delle impostazioni della simulazione da effettuare. In tale file, vengono settati:
  1. numero dei nodi della rete
  2. durata della simulazione
  3. lunghezza di un ciclo di simulazione
  4. protocolli e controlli utilizzati
  5. protocolli linkable e transport
  6. classe di modellazione per i nodi (se diversa da Node)
  7. momenti di esecuzione dei controlli di Peersim
  8. inizializzazione di parametri locali delle istanze dei controlli e protocolli utilizzati

  Un esempio di file di configurazione è il seguente:

```
random.seed 1234567890
simulation.cycles 30
control.shf Shuffle
network.size 50000

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache 20
protocol.avg example.aggregation.AverageFunction
protocol.avg.linkable lnk

init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k 20
init.pk example.aggregation.PeakDistributionInitializer
init.pk.value 10000
init.pk.protocol avg
init.ld LinearDistribution
init.ld.protocol 1
```

```
init.ld.max 100
init.ld.min 1

# you can change this to include the linear initializer instead
include.init rnd pk

control.ao example.aggregation.AverageObserver
control.ao.protocol avg

control.dnet DynamicNetwork
control.dnet.add -500
control.dnet.from 5
control.dnet.until 10
```

Per quanto riguarda i protocolli, PeerSim permette di utilizzare protocolli CycleDriven, i quali ad ogni ciclo eseguono un'azione che viene definita attraverso il metodo *nextCycle*(), e protocolli EventDriven, i quali modellano in un modo più realistico la rete poiché le azioni da eseguire per ogni nodo sono invocate alla ricezione di messaggi nella rete.

Nell'implementazione di ACEProtocol abbiamo utilizzato il modello basato su protocolli EventDriven, sia perché la suite di Protocolli ACE è event-driven, sia per motivi di confronto in fase di test con altri algoritmi.

5.2 Calcolo della triangolazione di Delaunay

Il calcolo della triangolazione di Delaunay locale per ogni nodo viene effettuato utilizzando la libreria esterna JTS (http://www.vividsolutions.com/jts/jtshome.htm). Il calcolo della triangolazione di Delaunay avviene utilizzando un'istanza della classe DelaunayTriangulationBuilder, la quale calcola l'insieme di archi che formano la triangolazione tramite il metodo *setSites(Collection<Coordinate>())*.

5.3 Classi utilizzate

Distinguiamo le classi utilizzate nel codice in:

1. *Managers*, utilizzati per gestire determinate funzioni dell'applicazione. Tra questi distinguiamo:

- il *TestManager*, il quale si occupa di memorizzare la mappa relativa alla relazione tra numero di messaggi e l'hitratio corrispondente, informazioni necessarie in fase di test di accuratezza dell'algoritmo.
- il *DelaunayCorrectnessManager*, il quale si occupa di controllare la correttezza della triangolazione di Delaunay distribuita tramite il metodo *checkNeighborsFiles(String oracoloFile, String neighborDistrFile)*, il quale riceve in input due file xml rappresentanti i vicini di un nodo n per l'oracolo e per l'algoritmo e restituisce il numero di differenze tra questi due file. Un esempio della struttura adottata dall'xml è:

```xml
<?xml version="1.0"?>
<Node Nodo="3">
  <Coordinates>809,4243</Coordinates>
  <Neighbours>
    <Neighbour>982,4024</Neighbour>
    <Neighbour>1045,4715</Neighbour>
    <Neighbour>769,4438</Neighbour>
    <Neighbour>717,4173</Neighbour>
    <Neighbour>661,4445</Neighbour>
  </Neighbours>
</Node>
```

- il *NearestNeighborCorrectnessManager*, utilizzato per verificare la correttezza del calcolo del nearest neighbor per ogni nodo nel caso in cui al tempo t in cui avviene questo calcolo la triangolazione di Delaunay non sia corretta (non garantendo quindi che il greedy routing funzioni).
- il *PointsManager*, il quale è incaricato di effettuare lo scanning del file dei nodi in input e crearne un Array di DTPoint. Tali punti verranno assegnati ai nodi della rete in fase di inizializzazione del protocollo InetInitializer.

2. *Inizializzatori*, protocolli con lo scopo di inizializzare la rete prima dell'esecuzione dell'algoritmo. L'unico inizializzatore utilizzato è *InetInitializer*, il quale ha lo scopo di assegnare ai vari nodi della rete le coordinate (X, Y) salvate dal PointsManager nel vettore di DTPoints utilizzando il protocollo *InetCoordinates*.

3. *Controlli*, utilizzati per effettuare determinate operazioni su tutti i nodi della rete a determinati cicli. In questo caso, ho utilizzato i controlli di PeerSim per implementare il churn della rete e l'inserzione dei nodi. Il controllo relativo all'inserimento dei nodi nella rete permette di settare una variabile di istanza per definire il numero di nodi che effettuano la fase di join in modo concorrente. Tale variabile, se settata a 1, definisce

un inserimento dei nodi sequenziale, con lo scopo di poter effettuare test su nodi che effettuano delle join sequenziali nella rete. I controlli utilizzati sono:

- il *NodeInsertionControl*, che memorizza la variabile $NUM_CONCURRENCY$ utilizzata per impostare il grado di concorrenza utilizzato per l'inserimento dei nodi nella rete, e finché sono presenti nuovi nodi da aggiungere alla rete permette a questi di effettuare la Join, inviando il messaggio opportuno al *bootstrapNode*.
- il *LeaveControl*, il quale si occupa della simulazione del churn della rete: esso permette l'uscita e l'entrata dei nodi all'interno della rete utilizzando le seguenti costanti:
  (a) $JOIN_PROBABILITY$, la quale definisce la probabilità che un nodo n non attivo nella rete ha di effettuare la join.
  (b) $LEAVE_PROBABILITY$, la quale definisce la probabilità che un nodo n arrivo nella rete ha di effettuare la leave.

  Inoltre, la classe LeaveControl permette il controllo del numero di nodi che escono dalla rete direttamente dal file di configurazione di PeerSim utilizzando la variabile privata $numNodesToLeave$.
- il *PeriodicTimer*, controllo che simula il timer del protocollo di manutenzione scaduto. Ogni volta che il metodo *execute*() di tale controllo viene invocato, il metodo *timerExpired*() della classe *ACEProtocol* viene eseguito.
- il *CheckDeadNodesControl*, il quale si occupa della simulazione del timer scaduto per i nodi ai quali si era fatta richiesta di vicini in fase di manutenzione.

4. *Osservatori*, i quali osservano determinati valori per i nodi della rete. Le classi implementate tra gli osservatori sono:

   - la classe *CoordinatesObs*, la quale in fase di inizializzazione stampa i valori delle coordinate dei nodi che effettueranno la join nella rete.
   - la classe *AccuracyObserver*, la quale si occupa di stampare nei files numMessagesFile.txt e accuracyValuesFile.txt i valori hitRatio e numero di messaggi memorizzati nelle strutture dati del TestManager. Tali files verranno utilizzati come raccolta dei dati per la stampa dei grafici in fase sperimentale.
   - la classe *NeighborsObserver*, che ha il compito di stampare su files xml i vicini di ogni nodo, i quali sono utilizzati dal DelaunayCorrectnessManager per effettuare il test di correttezza della triangolazione di Delaunay distribuita calcolata.

5. *Protocolli*, che implementano l'algoritmo vero e proprio. Questi saranno descritti nei prossimi paragrafi.

5.3.1 DTTransport e DDTNode

La classe *DTTrasport* implementa il protocollo *Transport* di PeerSim. In particolare, la classe *ReliableTransport* di PeerSim definisce un protocollo di trasporto *affidabile*, che comunque consente di impostare eventuali ritardi tramite valori numerici definiti nel file di configurazione.
La classe *DTTrasport* estende la classe che definisce il protocollo affidabile di trasporto di PeerSim, ed inoltre richiama l'istanza condivisa della classe *TestManager* per incrementare il contatore dei messaggi inviati ogni volta che il metodo *send*() viene invocato. Questo viene utilizzato in fase di test.
La classe *DDTNode* estende la classe *GeneralNode* di PeerSim e viene utilizzata per modellare i nodi della rete. Essa fornisce le interfacce pubbliche *compareTo*() e *isBootstrapNode*(), la prima utilizzata per confrontare due nodi e l'altra per verificare se il nodo corrente è il nodo di bootstrap (utilizzato in fase di join).

```java
public class DDTNode extends GeneralNode implements Comparable<DDTNode> {

    public DDTNode(String prefix)
    {
        super(prefix);
    }

    @Override
    public int compareTo(DDTNode o)
    {
         return(int)(getID() - o.getID());
    }

    public boolean isBootstrapNode()
    {
        return this.getID() == 0;
    }
}
```

Figura 5.1: Classe DDTNode.java

5.3.2 Messaggi

Per la simulazione è stata implementata una classe per ogni messaggio descritto all'interno del paper di Lee [21]. In più, sono stati implementati i nuovi messaggi descritti nel capitolo precedente per New ACE. In particolare, i messaggi implementati per l'algoritmo originale ACE sono:

- Delete
- JoinQuery
- JoinAnswer

- Leave
- NeighborNotification
- NeighborSetReply
- NeighborSetRequest
- Remove

I messaggi utilizzati nell'implementazione di New ACE sono:

- Delete
- JoinQuery
- JoinAnswer
- Leave
- NeighborReply
- NeighborRequest
- Acknowledge
- Remove

Al fine di evitare codice duplicato e utilizzare degli identificatori per determinare il tipo di messaggio ricevuto, tutte queste classi estendono la classe *AceEvent*, la quale rappresenta un messaggio generico utilizzato nell'algoritmo ACE e ne permette l'identificazione tramite l'utilizzo di un tipo enumerato. La figura 5.2 descrive l'implementazione della classe.

```java
package it.catania.ddt2d.Messages;

/*
 * this class is the superclass of all the messages used in the paper plus the JoinQuery and the
 * JoinAnswer message.
 */

public class AceEvent {

    public enum AceEventIdentifier{
        NeighborSetRequestId, NeighborSetReplyId, NeighborNotificationId,
        JoinQueryId, JoinAnswerId,
        LeaveId, DeleteId,
        ContingencyPlanId, PingId, PongId,
        RemoveId,
        NeighborRequestId, NeighborReplyId, AcknowledgeId
    }

    private AceEventIdentifier identifier;

    public AceEvent(AceEventIdentifier identifier)
    {
        this.identifier = identifier;
    }

    public AceEventIdentifier getIdentifier()
    {
        return identifier;
    }
    public void setIdentifier(AceEventIdentifier identifier)
    {
        this.identifier = identifier;
    }
}
```

Figura 5.2: La classe AceEvent

5.3.3 GreedyNearestNeighbor

La classe *GreedyNearestNeighbor* implementa il protocollo utilizzato per trovare il vicino più vicino dei nodi nella rete utilizzando l'algoritmo greedy. Attraverso tale protocollo sono inviati i messaggi *JoinQuery* e *JoinAnswer*. Le azioni che ogni nodo compie alla ricezione di tali messaggi sono definiti nei metodi privati *processJoinQueryEvent* e *processJoinAnsweEvent*.

Il metodo *processJoinQueryEvent*() è descritto di seguito.

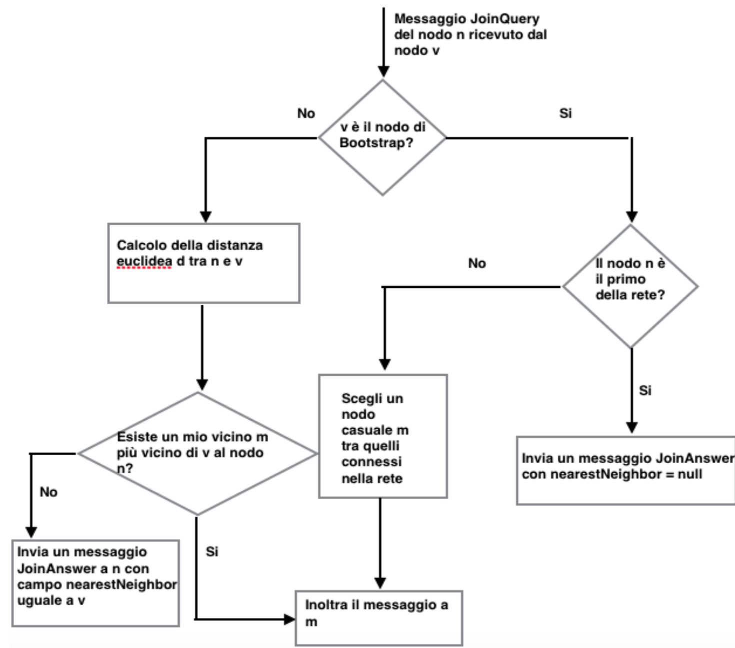

Figura 5.3: Diagramma di flusso alla ricezione del messaggio di JoinQuery per la classe GreedyNearestNeighbor

La Figura 5.3 descrive il comportamento della classe GreedyNearestNeighbor nel momento in cui viene invocato il metodo $processJoinQueryEvent()$: se il nodo contenitore del protocollo (che riceve il messaggio) è il nodo di bootstrap, viene simulato l'avvio dell'algoritmo greedy per trovare il nodo più vicino scegliendo un nodo a caso nella rete e inoltrandogli il messaggio $JoinQuery()$ ricevuto. Se non ci sono nodi nella rete (il nodo di bootstrap ne è consapevole), allora viene comunicato al nodo entrante che è il primo nodo della rete, mediante l'invio di un messaggio $JoinAnswer(null)$.
Se il nodo contenitore c del protocollo non è il nodo di bootstrap, allora cerca tra i suoi vicini un nodo v più vicino al nodo entrante: se esiste un nodo w più vicino di c a v, c inoltra il messaggio $JoinQuery()$ ricevuto a w, altrimenti significa che c è il nodo più vicino a v, quindi risponde con un messaggio $JoinAnswer(c)$.
Alla ricezione di un messaggio JoinAnswer, il nodo contenitore setta la propria vista locale e invia al nodo più vicino un messaggio $NeighborRequest()$ sul protocollo $ACEProtocol$.

5.3.4 ACELinkable

ACELinkable implementa la vista dei nodi della classe *DTNode* nella simulazione. Essa fornisce le seguenti istanze:

- *neighbors*, di tipo $ArrayList < DDTNode >$, utilizzata per memorizzare la lista dei vicini nella triangolazione di Delaunay. Da tale lista sono presi i dati per stampare i

files xml utilizzati in fase di test.

- $candidateSet$, di tipo $ArrayList < DDTNode >$, utilizzata per memorizzare la Candidate Set.
- $querriedNeighbors$, di tipo $ArrayList < DDTNode >$, non utilizzata in New ACE ma usata per effettuare dei confronti di accuratezza tra ACE e New ACE.
- $receivedAnswers$, di tipo $LinkedHashMap < DDTNode, Boolean >$, utilizzata per tener traccia dei nodi da cui si è ricevuta risposta dopo che è stato invocato il protocollo di manutenzione, in modo da eliminare eventuali nodi usciti dalla rete.

Oltre ai metodi getter e setter, la classe fornisce il metodo $getMutualNeighbors(Node\ w)$, il quale restituisce i nodi vicini in comune tra w e il nodo corrente.

5.3.5 ACEProtocol

La classe *ACEProtocol* implementa l'interfaccia *EDProtocol* di PeerSim e rappresenta l'implementazione dell'intero algoritmo ACE, poiché a parte i messaggi *JoinQuery()* e *JoinAnswer()* intercetta e gestisce tutti gli altri messaggi inviati.

```java
public void processEvent(Node node, int protocolID, Object event)
{
    AceEvent aceEvent = (AceEvent)event;

    switch(aceEvent.getIdentifier())
    {
        case NeighborSetRequestId:
            this.processNeighborSetRequestMessage((DDTNode)node, protocolID, (NeighborSetRequest)aceEvent);
            break;
        case NeighborSetReplyId:
            this.processNeighborSetReplyMessage((DDTNode)node, protocolID, (NeighborSetReply)aceEvent);
            break;
        case NeighborNotificationId:
            this.processNeighborNotificationMessage((DDTNode)node, protocolID, (NeighborNotification)aceEvent);
            break;
        case LeaveId:
            this.processLeaveMessageReceived((DDTNode)node, protocolID, (Leave)aceEvent);
            break;
        case DeleteId:
            this.processDeleteMessageReceived((DDTNode)node, protocolID, (Delete)aceEvent);
            break;
        case RemoveId:
            this.processRemoveMessageReceived((DDTNode)node, protocolID, (Remove)aceEvent);
            break;
        case NeighborRequestId:
            this.processNeighborRequestMessage((DDTNode)node, protocolID, (NeighborRequest)aceEvent);
            break;
        case NeighborReplyId:
            this.processNeighborReplyMessage((DDTNode)node, protocolID, (NeighborReply)aceEvent);
            break;
        case AcknowledgeId:
            this.processAcknowledgeMessage((DDTNode)node, protocolID, (Acknowledge)aceEvent);
            break;
        case NotNeighborId:
            this.processNotNeighborMessage((DDTNode)node, protocolID, (NotNeighbor)aceEvent);
            break;
        default:
            break;
    }
}
```

Figura 5.4: Implementazione del metodo *processEvent()*

La classe implementa i metodi che gestiscono la ricezione dei messaggi in ACE originale e New ACE. Tali metodi sono invocati tramite il metodo $processEvent()$, definito nell'interfaccia EDProtocol di PeerSim, il quale simula la ricezione di un messaggio tramite il protocollo di Trasporto utilizzato nella simulazione.
Oltre a questi metodi, la cui implementazione segue fedelmente la descrizione del Capitolo 3, la classe ACEProtocol fornisce una serie di metodi che vanno menzionati:

- il metodo $periodicTimerExpired()$, invocato dalla classe periodicTimer per la simulazione dello scadere del timer periodico che invoca l'esecuzione del protocollo di manutenzione;
- i metodi ausiliari privati che permettono di ricavare i nodi vicini di un nodo dalla triangolazione di Delaunay vista come un insieme di triangoli e un insieme di archi, tra i quali:
  - il metodo *computeNeighborsInDelaunayTriangulationForNodeAndPoints()*, con parametri u e *candidateSet*, il quale ha il compito di aggiornare la triangolazione di Delaunay locale del nodo u utilizzando *candidateSet* e di restituire la lista di vicini di u;
  - il metodo *computeSetOfTriangleThatContainsNButDontContainsAnyQuerriedNode()*, con parametri u e $querriedNodes$, il quale ha il compito di calcolare la lista di triangoli che contengono il nodo u ma non contengono alcun nodo nella lista $querriedNodes$

5.3.6 InetCoordinates

La classe InetCoordinates rappresenta il protocollo che gestisce le coordinate geografiche assegnate ai nodi: in particolare, le variabili di istanza implementate sono:

- la variabile x, di tipo BigInteger, per la memorizzazione della latitudine associata al nodo contenitore;
- la variabile y, di tipo BigInteger, per la memorizzazione della longitudine associata al nodo contenitore.

5.4 Implementazione del protocollo di manutenzione

Le componenti principali che implementano il protocollo di manutenzione sono:

- la classe *PeriodicTimer*, che ha il compito di lanciare il protocollo di manutenzione su tutti i nodi effettuando un ciclo su questi invocando il metodo *periodicTimerExpired()*

implementato nella classe *ACEProtocol*. In dettaglio, la classe PeriodicTimer implementa la funzionalità principale del protocollo di Manutenzione, cioè l'invio di nuove richieste di vicinato.

- la classe *CheckDeletedNodesControl*, che ha il compito di controllare i nodi che non sono più attivi nella rete. Ogni volta che viene eseguito tale controllo, ogni nodo della rete verifica se la mappa *receivedAnswers* nella propria interfaccia Linkable contenga il valore *true* per ogni nodo a cui si è inviato un messaggio in fase si esecuzione del protocollo di manutenzione:
  - se non lo contiene, viene eseguito il metodo che implementa la *Remove* del nodo
  - se lo contiene, non viene fatto nulla.

Affinché il protocollo di manutenzione sia simulato correttamente, è necessario che il controllo PeriodicTimer e CheckDeletedNodesControl siano sincronizzati nel loro tempo di esecuzione, con il controllo CheckDeletedNodesControl eseguito un tempo δ dopo il periodicTimer. Nell'implementazione effettuata $\delta = 35$ cicli di PeerSim.

Capitolo 6

Risultati sperimentali

Lo scopo di questo capitolo è quello di presentare i principali risultati sperimentali ottenuti dall'esecuzione di alcuni test per misurare e confrontare le performance dei protocolli analizzati nel capitolo 4.

Oltre a misurare le effettive performance dei protocolli, i risultati ottenuti consentono di confrontare i protocolli ACE, New ACE e GoDel. Essendo GoDel oggetto di lavori precedenti a questa tesi, è stata utilizzata la precedente implementazione per i risultati descritti in questo capitolo.

I test sono focalizzati nel determinare, per i protocolli, il numero di cicli di simulazione necessari per la costruzione di una triangolazione corretta e il relativo numero di messaggi utilizzato per il raggiungimento di questo risultato.
Una convergenza alla soluzione corretta con un numero di messaggi limitato è importante in applicazioni sviluppate su reti che possiedono requisiti di banda limitata e di consumo di energia, per esempio alle reti di sensori.
Il capitolo presenta inoltre una serie di test aggiuntivi effettuati sui protocolli ACE e New ACE, al fine di fornire una prova sperimentale delle considerazioni effettuate sulla correttezza di ACE e New ACE nel capitolo 4, e al fine di misurare le performance di questi.

Per la definizione dell'accuratezza della triangolazione di Delaunay calcolata è stato utilizzato un *Oracolo* interno al codice di ACE e New ACE, il quale ha lo scopo di costruire la triangolazione di Delaunay sull'insieme dei nodi partecipanti alla rete e di stampare il risultato in files xml dedicati, al fine di poterli utilizzare per il confronto con i files dei vicini dei nodi calcolati con gli algoritmi distribuiti.

La misura dell'accuratezza della triangolazione calcolata è definita come *hitratio* (6.1) ed è il rapporto tra il numero totale di vicini corretti dei nodi nella triangolazione calcolata in maniera distribuita e il numero di vicini totale dei nodi nella triangolazione calcolata dall'Oracolo.

$$hitratio = \frac{\#CorrectDistribuitedNeighbors}{\#OracoloNeighbors} \tag{6.1}$$

Per gli esperimenti eseguiti sono stati utilizzati due DataSets diversi:

1. un *DataSet sintetico*, generato con una distribuzione uniforme, composto da 2000 nodi ed utilizzato principalmente per i test di confronto tra ACE e New ACE. Al fine di valutare reti di diversa dimensione, per i test sono stati utilizzati sottoinsiemi di 500,1000,1500 e 2000 nodi.

2. un *DataSet reale*, composto da 500 coordinate su di un piano 5000x5000, che rappresentano un sottoinsieme delle Vivaldi Coordinates presenti all'interno del dataset originale [5], utilizzato per i test di confronto tra ACE, New ACE e GoDel.

6.1 Convergenza di ACE e New ACE per Join Sequenziali

Questa sezione descrive l'esperimento effettuato per confrontare la convergenza a una triangolazione di Delaunay corretta di ACE e New ACE nel caso di Join sequenziali, ovvero in cui una join inzia la sua esecuzione quando la precedente join è terminata. Si assuma, quindi, che i nodi eseguano la fase di join sequenzialmente e che non ci siano disconnessioni volontarie o involontarie di nodi. Come descritto nella sezione 4.3, il protocollo ACE non sempre converge a una soluzione corretta a causa di alcune assunzioni fatte nella dimostrazione del Lemma 4.2.2, di cui è stata provata la non completa veridicità tramite l'esempio nella sezione 4.3. Lo scopo di questo test è quello di verificare come tale assunzione errata incida sulla correttezza della triangolazione di Delaunay calcolata in relazione al numero di iterazioni in cui i nodi effettuano delle operazioni di join sequenzialmente e verificare la correttezza di NewACE.

L'esperimento è stato effettuato utilizzando un sottoinsieme di 500, 1000, 1500 e 2000 nodi del dataset sintetico, con lo scopo di valutare la scalabilità degli algoritmi analizzati.

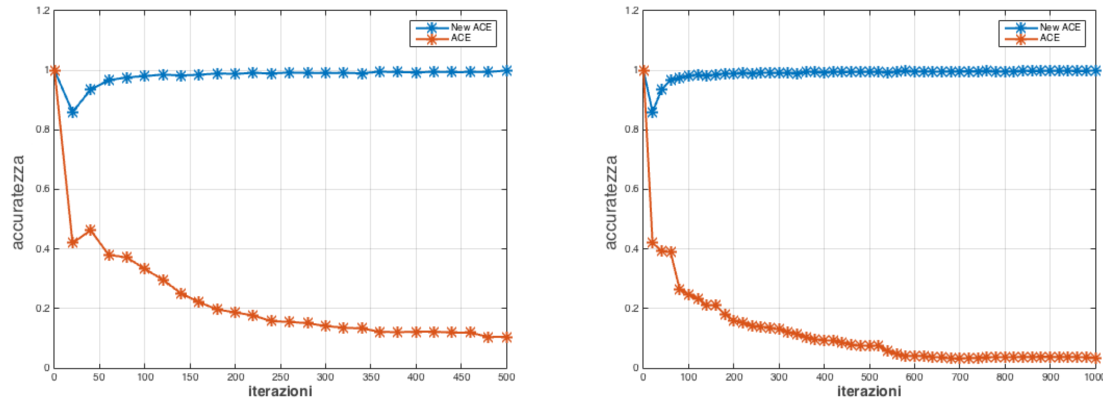

Figura 6.1: Convergenza di ACE e New ACE per 500 e 1000 nodi

La Figura 6.1 mostra i grafici ottenuti dall'esecuzione dell'esperimento su un insieme di nodi generato da i primi 500 (a sinistra) e 1000 (a destra) punti del dataset sintetico. Nel grafico è visibile che l'accuratezza della triangolazione di Delaunay costruita dal protocollo ACE diminuisce già dopo 20 iterazioni di più del 50%. Inoltre, questa continua a scendere all'aumentare del numero di cicli di simulazione, e quindi all'aumentare della dimensione della rete. Questo è motivato dal fatto che non è garantito che un nodo in fase di join trovi il nodo più vicino nella rete, perché la triangolazione in quel momento non è corretta, e questo da luogo ad un processo in cui le inconsistenze aumentano via via che si aggiungono nuovi nodi. New ACE, invece, dopo un primo picco di inconsistenza dovuto probabilmente a messaggi non ancora pervenuti, raggiunge l'accuratezza del 100%. In questo modo abbiamo dato una prova sperimentale della correttezza di New ACE per operazioni di join sequenziali.

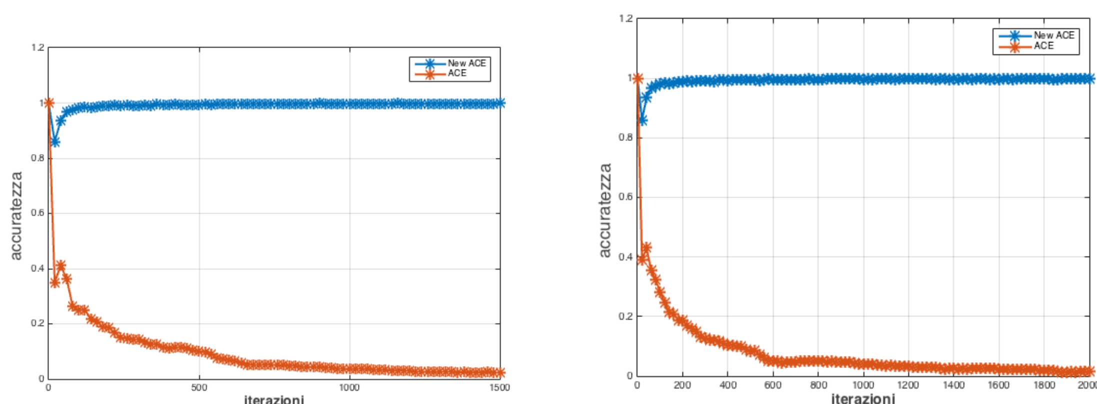

Figura 6.2: Convergenza di ACE e New ACE per 1500 e 2000 nodi

La Figura 6.2 mostra l'esperimento eseguito su reti, rispettivamente, di 1500 e 2000 nodi. Dai quattro esperimenti si evince che l'accuratezza finale di ACE dipende dal numero di nodi

nella rete: infatti, il grafico relativo all'esperimento eseguito su 500 iterazioni in cui 500 nodi entrano sequenzialmente mostra per ACE un accuratezza maggiore rispetto al grafico relativo allo stesso esperimento eseguito su 2000 iterazioni in cui 2000 nodi entrano sequenzialmente. Al contrario, l'accuratezza di New ACE rimane stabile sul 100%.

6.2 Numero di messaggi per Join Sequenziali

Il secondo esperimento è stato effettuato per verificare il numero di messaggi inviati da ogni nodo in relazione ai cicli di simulazione in cui, ad ogni ciclo di simulazione, un nodo esegue la fase di join. Come nella sezione precedente, si assuma che i nodi eseguano la fase di join sequenzialmente e che non ci siano disconnessioni volontarie o involontarie di nodi. L'esperimento è stato effettuato su un sottoinsieme di 500, 1000, 1500 e 2000 punti del dataset sintetico.

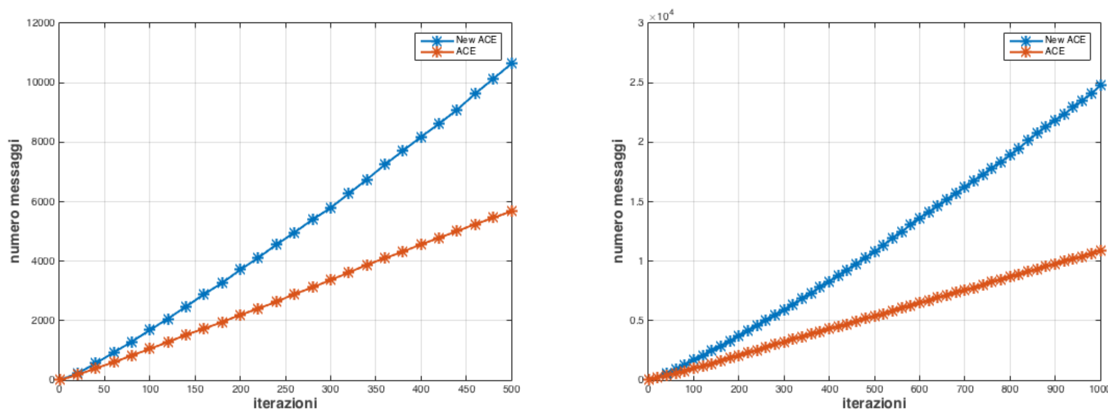

Figura 6.3: Messaggi di ACE e New ACE per 500 e 1000 nodi

La Figura 6.3 mostra il numero di messaggi inviati in relazione al numero di cicli di simulazione per 500 join sequenziali (a sinistra) e per 1000 join sequenziali (a destra). Nei due grafici è in evidenza che l'ottimizzazione per la riduzione del numero di messaggi in fase di join di ACE funziona, poiché questo invia in media un numero molto inferiore di messaggi rispetto a New ACE. Purtroppo, come verificato nell'esperimento descritto nella sezione 7.1, il basso numero di messaggi di ACE porta alla costruzione di una triangolazione non corretta a causa delle considerazioni già effettuate nella sezione 7.1. Il grafico mostra, inoltre, che il numero di messaggi per ACE è relativamente basso per il problema da risolvere.

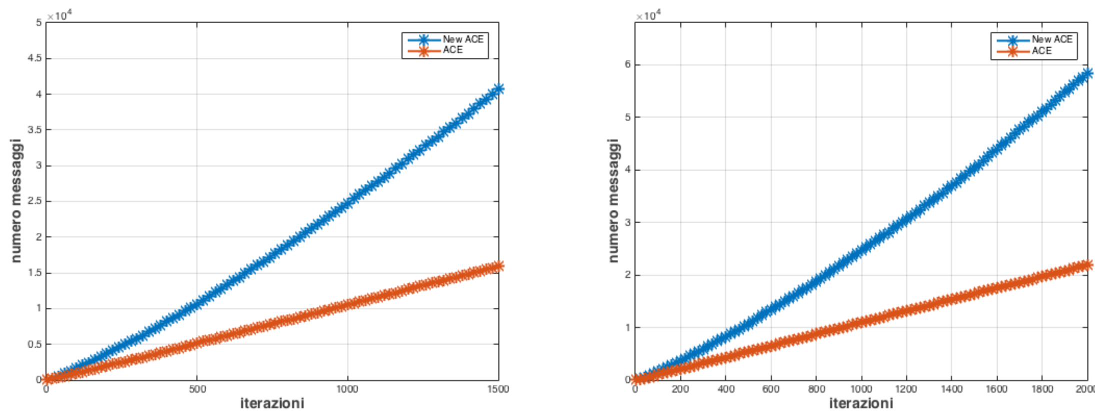

Figura 6.4: Messaggi di ACE e New ACE per 1500 e 2000 nodi

La Figura 6.4 mostra lo stesso test effettuato su ACE e New ACE per 1500 e 2000 entrate sequenziali, in 1500 e 2000 cicli di simulazione. Anche in questo caso, il numero di messaggi utilizzato da ACE è inferiore a quello utilizzato da New ACE, con la differenza che New ACE converge a una soluzione corretta, come mostrato dai grafici in Figura 6.2.

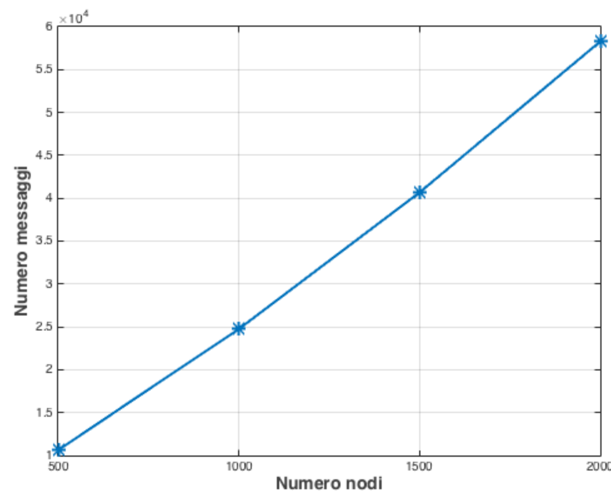

Figura 6.5: Numero messaggi utilizzati in New ACE

La Figura 6.5 mostra la relazione tra il numero di messaggi utilizzati dal protocollo New ACE in relazione al numero di entrate sequenziali nella rete. In dettaglio, la curva nel grafico mostra che tale relazione si avvicina molto alla linearità.

I test effettuati sono stati analizzati in maniera più approfondita. Nell'analisi effettuata, si è cercata una relazione possibile tra il numero di nodi della rete e il numero di messaggi necessari alla convergenza di New ACE utilizzando il Teorema 2.1.4.

Numero nodi	Numero di messaggi teorizzato	Numero di messaggi sperimentale	δ medio sperimentale
500	$500\times(15+\delta)$	10638	6
1000	$1000\times(15+\delta)$	24753	9
1500	$1500\times(15+\delta)$	40680	12
2000	$2000\times(15+\delta)$	58288	14

Tabella 6.1: Numero di messaggi teorizzato e sperimentale per test sulle Join sequenziali dei nodi

Sia n il numero di nodi nella rete e supponiamo per ipotesi che il numero di vicini per ogni nodo della rete sia 6. Questa ipotesi è ragionevole in quanto, come discusso nel paragrafo 2.1, il numero medio di vicini di un nodo in una rete di Delaunay è proprio 6 (Teorema 2.1.4). Supponiamo inoltre che al momento della join la rete sia stabile.

1. nella prima fase di join il nodo che ha inviato il primo messaggio viene a conoscenza tramite il protocollo greedy del suo vicino più vicino. Questo comporta l'invio di 1 messaggio al nodo di bootstrap, il quale provvederà a indirizzare tale messaggio a un nodo a caso della rete, inviano 1 altro messaggio. Da qui in poi, saranno inviati δ nuovi messaggi per trovare il nodo destinatario del messaggio, il quale risponderà al nodo che effettua al join con 1 nuovo messaggio. Quindi, il numero di messaggi inviati per la prima fase di join è
$$nmessages = 3 + \delta \tag{6.2}$$
2. nella seconda fase di join, vengono scambiati 2 messaggi (uno di richiesta e uno di risposta) per ogni vicino conosciuto. Utilizzando l'ipotesi che il numero di vicini per nodo sia 6, nella seconda fase di join ogni nodo invia 12 messaggi.

La Tabella mostra il valore di δ teorico ricavato dagli esperimenti, il quale rappresenta il numero di messaggi inviati in fase greedy routing da ogni nodo.

6.3 Leave sequenziali

L'esperimento descritto in questa sezione ha lo scopo di verificare la relazione tra il numero di nodi che effettuano una disconnessione volontaria dalla rete e il numero di messaggi inviato di conseguenza.

L'esperimento è stato effettuato solamente sul protocollo New ACE poiché, come descritto nella sezione 6.1, ACE non converge a una soluzione corretta in caso di join sequenziali. In dettaglio, dopo la stabilizzazione della rete, per 50 cicli di simulazione, ad ogni iterazione, un nodo scelto casualmente decide di disconnettersi con una probabilità p. Al fine di avere un

Probabilità di disconnessione	Numero disconnessioni	Numero di messaggi
$p = 25\%$	18	10266
$p = 50\%$	25	10098
$p = 75\%$	36	18031
$p = 100\%$	48	19244

Tabella 6.2: Relazione tra disconnessioni volontarie e numero di messaggi inviati

diverso numero di nodi che effettua la disconnessione, ad ogni test è associata una probabilità p diversa di uscita dei nodi, variando tra:

- $p = 25\%$
- $p = 50\%$
- $p = 75\%$
- $p = 100\%$

Nel test effettuato è stato deciso di non mostrare il grafico relativo all'accuratezza della leave sequenziale perché, come nel caso del test relativo alle join sequenziali, l'accuratezza è stabile sul 100%. La tabella 6.2 mostra il numero di nodi che hanno effettuato la disconnessione, la probabilità di uscita dei nodi data dalla rete e il numero di messaggi inviato nella rete.

Dalla Tabella 6.2 si evince che non esiste una relazione forte tra il numero di nodi che effettuano una disconnessione e il numero di messaggi inviati per la sua gestione, poiché questa dipende da altri fattori, come la posizione e il numero di nodi posizionati nell'intorno del nodo che effettua la disconnessione.

6.4 Test su Dataset Reale con churn

Lo scopo dei test presentati in questa sezione è quello di effettuare un confronto tra ACE, New ACE e GoDel utilizzando un DataSet reale di 500 nodi contenente un insieme di Vivaldi network coordinates [5]. L'implementazione di GoDel utilizzata per la fase di test è relativa a lavori precedenti e non è quindi stata effettuata una sua nuova implementazione.

Gli esperimenti effettuati hanno lo scopo di confrontare il numero di cicli di simulazione necessari a ACE, New ACE e GoDel necessari alla convergenza e il numero di messaggi utilizzati in questi. Tutti i test prevedono che i 500 nodi generati dal DataSet di Vivaldi network coordinates effettuino la join in maniera concorrente. Per questo motivo, ACE e New ACE prevedono l'esecuzione del protocollo di manutenzione. Nell'esperimento osservato, il protocollo di manutenzione viene eseguito ogni 50 cicli di simulazione.

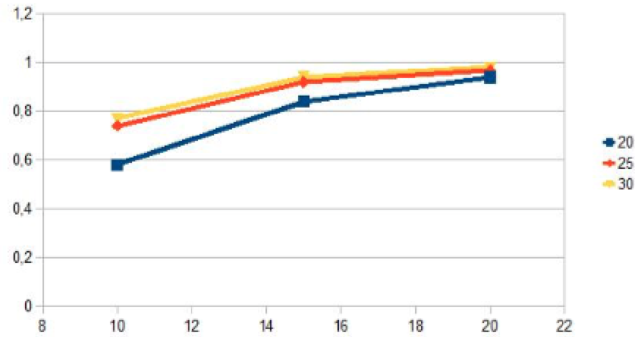

Figura 6.6: Convergenza di GoDel al variare della vista scambiata dai nodi

In GoDel il test è stato eseguito 3 volte variando la dimensione della vista locale scambiata dai nodi in accordo al protocollo di gossip Cyclon. Come mostrato in Figura 6.6, per tutte e tre le dimensioni della vista locale dei nodi la convergenza si raggiunge in soli 20 cicli di simulazione.

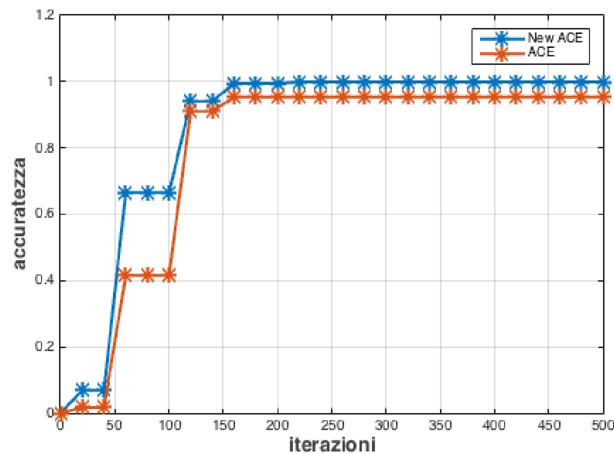

Figura 6.7: Convergenza di ACE e New ACE

La Figura 6.7 mostra invece la convergenza dei protocolli ACE e New ACE utilizzando il DataSet reale. Dal grafico sono evidenti per entrambi i protocolli intervalli di iterazioni in cui l'accuratezza entra in stallo. Questi intervalli sono motivati dal fatto che sia ACE che New ACE devono attendere l'esecuzione del protocollo di manutenzione, che avviene ogni 50 cicli di simulazione, per la riparazione della rete dovuta all'entrata dei nodi in maniera concorrente.

Di particolare interesse è la valutazione del numero di messaggi inviati per la costruzione

della triangolazione. Per GoDel è stato verificato che il numero di messaggi in relazione ai cicli di simulazione è definito dalla formula:

$$\#messages = 4 \times \#nodes \times \#iterations \tag{6.3}$$

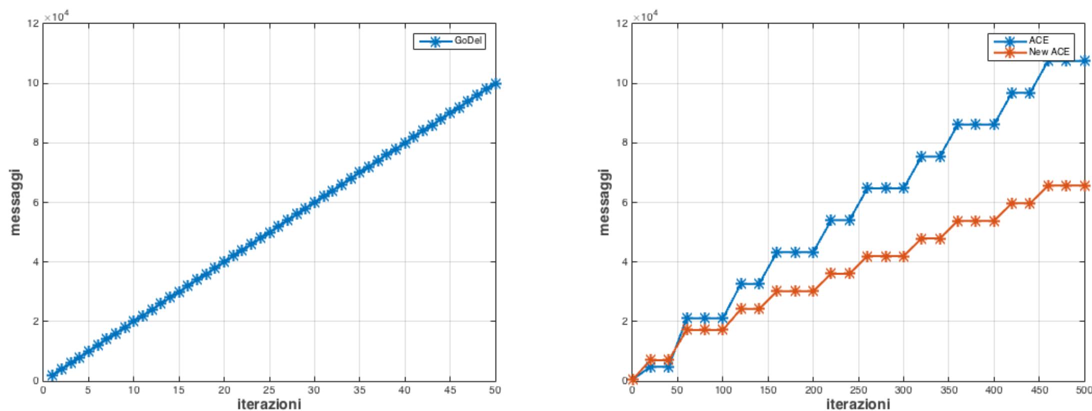

Figura 6.8: Crescita del numero di messaggi in GoDel, ACE e New ACE

La Figura 6.8 mostra il numero di messaggi in relazione ai cicli di simulazione per GoDel (a sinistra), ACE e New ACE (a destra) in una rete con 500 nodi. GoDel presenta una crescita lineare rispetto al numero di iterazioni dovuta al fatto che i nodi non smettono mai di contattarsi grazie al protocollo di Gossip Cyclon, mentre ACE e New ACE presentano numerosi stalli dovuti al fatto che, dopo che il protocollo di join è terminato per tutti i nodi, bisogna attendere l'esecuzione del protocollo di manutenzione che richiede di nuovi messaggi nella rete. Dal grafico si evince che anche in New ACE, nonostante la fase iniziale in cui i messaggi sono maggiori di ACE, il numero di messaggi tenda a diventare più piccolo con l'aumentare dei cicli di simulazione.

Unendo i risultati delle figure 6.8, 6.7 e 6.6, possiamo stabilire che il numero di messaggi per la costruzione della triangolazione corretta in New ACE è inferiore rispetto a quello di GoDel, poiché il primo costruisce la triangolazione in circa 150 cicli di simulazione con circa 30000 messaggi, contro i 20 cicli di simulazione e i 40000 messaggi di GoDel. ACE, invece, non costruisce una triangolazione di Delaunay corretta, ma arriva a ottenere un accuratezza di circa il 94%.

Un altro esperimento effettuato ha lo scopo di verificare la resistenza dei protocolli New ACE e GoDel in relazione a fallimenti concorrenti di nodi nella rete. In dettaglio, dopo la stabilizzazione della rete (200 cicli di simulazione per New ACE), in un ciclo di di simulazione una percentuale p di nodi della rete effettua una disconnessione non volontaria. Le due esecuzioni del test prevedono la disconnessione del:

1. 10% dei nodi della rete;

2. 20% dei nodi della rete.

Al termine della disconnessione dei nodi, New ACE esegue il protocollo di manutenzione per 500 cicli di simulazione, al fine di riparare i gli errori nella rete dovuti alla concorrenza. GoDel, invece, non ha bisogno dell'esecuzione di un protocollo di manutenzione, in quanto la rete si auto-stabilizza grazie al gossip.

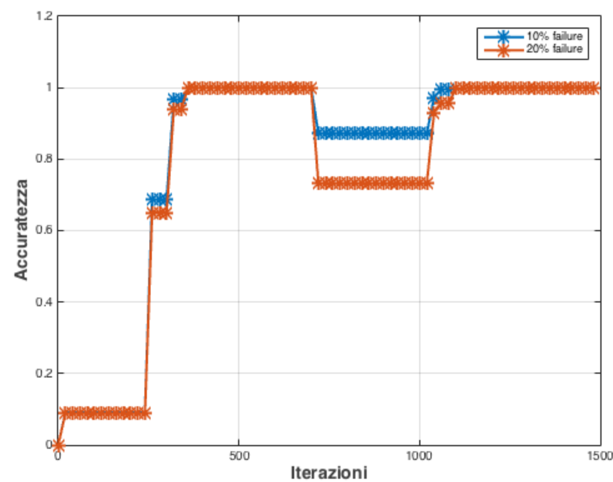

Figura 6.9: New ACE: accuratezza con churn di rete

Il grafico in Figura 6.9 mostra il test descritto precedentemente eseguito su New ACE per la disconnessione non volontaria del 10% e del 20% dei nodi della rete. Dal grafico si notino gli intervalli di cicli di simulazione in cui l'accuratezza rimane in stallo dovuti al problema di attesa di esecuzione del protocollo di manutenzione. Dopo l'avvio dell'esecuzione del protocollo di manutenzione, la rete torna corretta in circa 100 iterazioni, utilizzando circa 19000 messaggi.

Numero nodi usciti	Numero messaggi New ACE per stabilizzazione	Numero messaggi GoDel per stabilizzazione
50 (10%)	18577	$45 \times 4 \times 450 = 81000$
100 (20%)	19356	$45 \times 4 \times 400 = 72000$

Tabella 6.3: Numero di messaggi necessari a New ACE e GoDel per ricostruire la triangolazione corretta

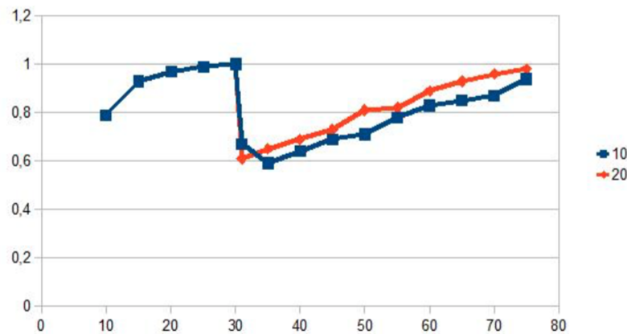

Figura 6.10: GoDel: accuratezza con churn di rete

La Figura 6.10 mostra lo stesso test eseguito su GoDel: nel grafico è particolarmente evidente che GoDel si auto-ripara, poiché non presenta dei plateau nella curva rappresentata e la sua accuratezza non smette mai di crescere. Questo torna corretto dopo appena 45 cicli di simulazione. La tabella 6.3 mostra il numero di messaggi necessario ai due protocolli nei test eseguiti per ristabilire la corretta triangolazione di Delaunay.

6.5 Test sul tempo di convergenza

In questa sezione sono riportati i risultati del test sul tempo richiesto per convergere a una triangolazione di Delaunay corretta per New ACE e GoDel. Il tempo di convergenza è misurato come il numero di iterazioni necessarie nella simulazione per la convergenza alla soluzione corretta. A differenza di GoDel, è stato verificato che il numero di iterazione necessarie ad ACE e New ACE per la costruzione di una soluzione corretta sia strettamente dipendente dal numero di esecuzioni del protocollo di manutenzione, quindi dalla frequenza dell'esecuzione di tale protocollo.

Al fine di provare ciò, sono stati effettuati alcuni test per ACE e New ACE utilizzando i 500 nodi del dataset reale variando la frequenza dell'esecuzione del protocollo per valutare il numero di iterazioni necessarie alla convergenza.

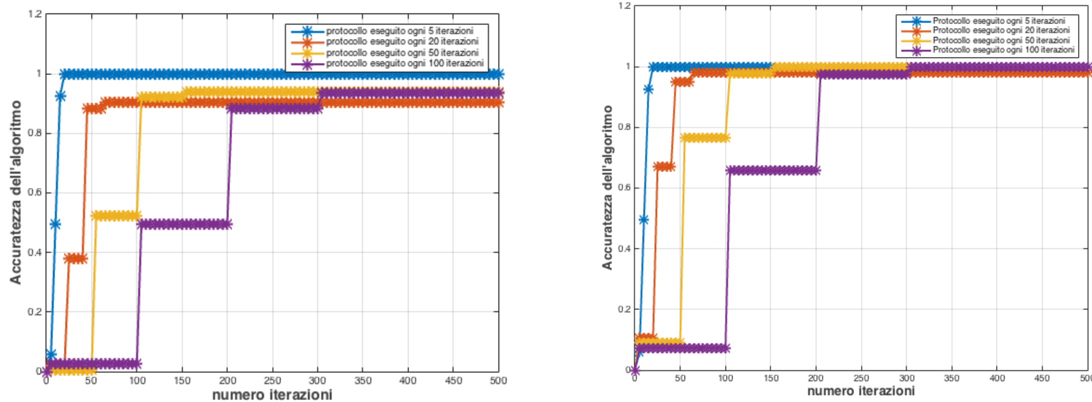

Figura 6.11: Convergenza di ACE (a sinistra) e new ACE (a destra)

La Figura 6.11 mostra la dipendenza del numero di cicli di simulazioni necessari per la convergenza a una triangolazione corretta dalla frequenza di esecuzione del protocollo di manutenzione per ACE (a sinistra) e New ACE (a destra). I grafici mostrano dei comportamenti molto simili per i due protocolli, e questo è dato dal fatto che fanno uso di un servizio temporizzato per la manutenzione. Nell'esperimento, per entrambi i protocolli ACE e New ACE il protocollo di manutenzione è stata eseguito ogni 5, 20, 50 e 100 cicli di simulazione. Dalla Figura 6.11 si evince che un protocollo di manutenzione eseguito frequentemente permette a ACE e New ACE di convergere in un numero relativamente piccolo di cicli di simulazione, mentre un protocollo di manutenzione eseguito meno frequentemente da luogo a una convergenza molto più lenta degli algoritmi.

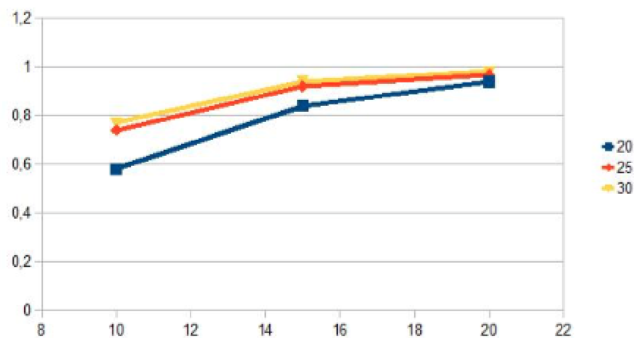

Figura 6.12: Convergenza di GoDel in relazione ai cicli di simulazione

La Figura 6.12 mostra il test effettuato su GoDel al variare della dimensione della vista locale scambiata tra i nodi in fase dal protocollo Cyclon (20,25,30). Il risultato è che in tutti e 3 gli esperimenti GoDel riesce a raggiungere la convergenza a una triangolazione di Delaunay corretta in 20 cicli di simulazione. È importante notare, inoltre, che in GoDel l'accuratezza continua a crescere ad ogni ciclo di simulazione grazie al gossip, il quale non rimane in stallo per intervalli di cicli di simulazione. Questo è un pregio di GoDel perché non richiede la determinazione della temporizzazione più opportuna per l'esecuzione del protocollo di manutenzione, cosa che può risultare difficile e dipendente dalle particolarità della rete di Delaunay. Infatti, se da un lato l'esecuzione più frequente del protocollo di manutenzione per ACE e New ACE permette il calcolo di una soluzione corretta in meno iterazioni, dall'altro incrementa il numero di messaggi utilizzato.

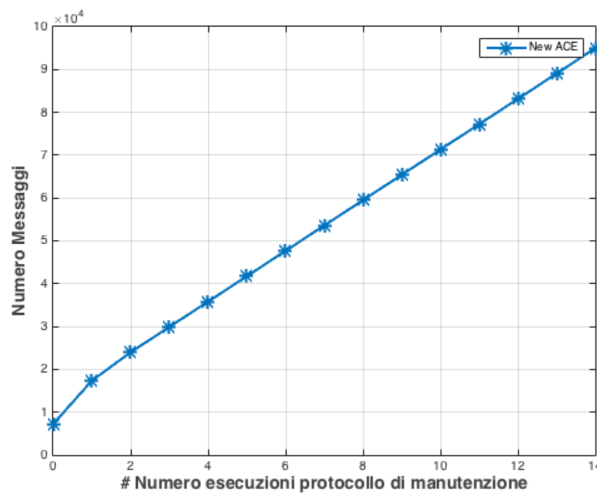

Figura 6.13: New ACE: numero di messaggi rispetto al numero di esecuzioni del protocollo di Manutenzione

La Figura 6.13 mostra il numero di messaggi utilizzati da New ACE rispetto al numero di esecuzioni del protocollo di manutenzione. In Figura, il numero di messaggi cresce linearmente rispetto al numero di esecuzioni del protocollo di manutenzione, quindi rispetto alla sua frequenza. Quindi, un'esecuzione molto frequente di questo protocollo renderebbe la rete di fatto non utilizzabile in numerose applicazioni reali a causa dell'elevato numero di messaggi, ma un'esecuzione meno frequente del protocollo potrebbe potrebbe non portare ad una soluzione corretta.

6.6 Considerazioni finali

Nelle sezioni precedenti sono stati presentati i risultati di alcuni esperimenti sui tre protocolli ACE, New ACE e GoDel. Tali risultati mettono in evidenza che GoDel si adatta molto bene alla dinamicità della rete dovuta a disconnessioni e connessioni dei nodi, anche con concorrenza, grazie al livello di gossip, che tramite lo scambio continuo della vista locale dei nodi permette l'auto-stabilizzazione della rete.

ACE e New ACE, in presenza di join concorrenti, non convergono a una soluzione corretta a causa della fase di greedy routing iniziale, la quale prevede che la triangolazione nel momento in cui si effettua il routing sia di Delaunay. Solo con il protocollo di manutenzione l'accuratezza raggiunge il 100%, ma il numero di cicli di simulazione necessario per raggiungerla è correlato al numero di esecuzioni del protocollo di manutenzione. Questo significa che i protocolli ACE e New ACE, a differenza di GoDel, richiedono una opportuna valutazione della frequenza di esecuzione del protocollo di manutenzione, al fine di trovare il giusto compromesso tra numero di messaggi nella rete e correttezza della triangolazione. Al contrario, GoDel è migliore da questo punto di vista grazie alla capacità di auto-stabilizzarsi. Altri test sono stati effettuati per confrontare il numero di messaggi inviati dai protocolli per raggiungere l'accuratezza del 100%. GoDel, a causa del gossip, invia un numero maggiore di messaggi rispetto a New ACE, ma comunque abbastanza ridotto.

In definitiva, il confronto ha portato alla esaltazione dei pregi e dei difetti dei due approcci:

1. il protocollo gossip, soffre di un alto numero di messaggi;

2. i protocolli che richiedono temporizzazione, che richiedono controlli esterni sulla temporizzazione e presentano periodi in cui la rete risulta instabile, in attesa della esecuzione del protocollo di sincronizzazione.

Capitolo 7

Conclusioni

Lo scopo di questa tesi è stato quello di analizzare l'insieme dei protocolli distribuiti per la costruzione di triangolazioni di Delaunay nell'ambito, principalmente, delle reti di sensori e delle reti P2P.

La triangolazione di Delaunay rappresenta una struttura molto interessante per modellare overlay con riferimenti geografici, ovvero ambienti in cui i peers sono individuabili attraverso delle coordinate in uno spazio d-dimensionale.

Nel capitolo 2 sono state descritte le definizioni e le principali proprietà matematiche della triangolazione di Delaunay. Inoltre, sono stati presentati i principali algoritmi proposti in letteratura per la costruzione di questa, considerando approcci incrementali e Divide et Impera.

Nel capitolo 3 è stata effettuata una rassegna dei principali algoritmi distribuiti per la costruzione della triangolazione di Delaunay, i quali sono stati raggruppati a seconda della loro applicazione. Tra questi, sono stati descritti algoritmi applicabili alle reti di sensori wireless, analizzando le problematiche dovute alla limitazione data dal raggio di trasmissione dei sensori. Inoltre, è stato considerata la relazione tra il numero di messaggi per la costruzione della struttura e l'impatto energetico sui sensori. Sono stati analizzati algoritmi di supporto al routing geografico, i quali utilizzano topologie basate su triangolazione di Delaunay aumentate con archi a lungo raggio. Inoltre, sono stati analizzati algoritmi orientati agli ambienti virtuali distribuiti: è stato descritto come, in questo scenario, debba essere posta particolare attenzione al numero di messaggi inviato, a causa della necessità di ricevere aggiornamenti in tempo reale per tutti i nodi della rete, con l'obiettivo di garantire la consistenza della visione dell'ambiente per tali nodi.

L'analisi quindi si è focalizzata nel capitolo 4 su due protocolli presenti in letteratura, con caratteristiche diverse, che risultano essere interessanti in quanto l'esecuzione degli stessi garantisce una costruzione totale dell'overlay (e non approssimata come in altri algoritmi analizzati).

Il primo protocollo preso in esame è stato GoDel, il quale utilizza protocolli di gossip per il riempimento delle viste locali di Delaunay dei nodi. GoDel rappresenta una scelta innovativa, poiché riduce notevolmente il numero di messaggi e garantisce la consistenza della rete. Essendo GoDel oggetto di un lavori precedenti, durante la tesi è stato valutato facendo uso della versione precedentemente implementata.

Il secondo protocollo è ACE, il quale utilizza l'approccio basato su Candidate Set per la costruzione della vista di Delaunay dei nodi. Il protocollo ACE prevede 4 procedure: una procedura per la fase di Join, una per la fase di Leave, una per la gestione dei Fallimenti e un protocollo di manutenzione per la correzione delle inconsistenze della rete dovute a operazioni concorrenti. Come GoDel, anche ACE riduce notevolmente il numero di messaggi e garantisce la costruzione totale dell'overlay di Delaunay.

Nella tesi è stato individuato che il protocollo non risulta corretto, è stata individuata la motivazione della non correttezza e attraverso l'introduzione di nuovi messaggi e di una nuova procedura per la Join è stato definito un nuovo protocollo, New ACE, basato su ACE, ma corretto.

I risultati sperimentali hanno consentito di confrontare i 3 protocolli e di valutare che l'approccio utilizzato da GoDel risulta essere migliore in quanto la rete si auto-stabilizza grazie al supporto del gossip. Lo svantaggio per GoDel è il numero di messaggi, che risulta essere ridotto in ACE e New ACE. Al contrario, ACE e New ACE hanno il vantaggio di un numero basso di messaggi per la convergenza ad una triangolazione corretta, ma lo svantaggio di richiedere l'esecuzione del protocollo di Manutenzione per la stabilizzazione della rete, il quale incrementa linearmente il numero di messaggi nella rete.

La tesi prevede una serie di sviluppi futuri: in particolare, si prevede di verificare formalmente la correttezza di New ACE. Inoltre, si prevede di ottimizzare GoDel grazie ai risultati ottenuti e di valutare ulteriori modifiche al protocollo New ACE, che permettano di eliminare l'esecuzione di un protocollo di Manutenzione, mantenendo ridotto il numero di messaggi.

Bibliografia

[1] N. Amenta, D. Attali e O. Devillers. «Size of Delaunay Triangulation for Points Distributed over Lower-dimensional Polyhedra: a Tight Bound». In: ().

[2] Filipe Araújo e Luís E. T. Rodrigues. «Fast Localized Delaunay Triangulation». In: *Principles of Distributed Systems, 8th International Conference, OPODIS 2004, Grenoble, France, December 15-17, 2004, Revised Selected Papers.* A cura di Teruo Higashino. Vol. 3544. Lecture Notes in Computer Science. Springer, 2004, pp. 81–93.

[3] Filipe Araújo e Luís E. T. Rodrigues. «GeoPeer: A Location-Aware Peer-to-Peer System». In: *3rd IEEE International Symposium on Network Computing and Applications (NCA 2004), 30 August - 1 September 2004, Cambridge, MA, USA.* IEEE Computer Society, 2004, pp. 39–46.

[4] Franz Aurenhammer. «Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure». In: *ACM Comput. Surv.* 23.3 (1991), pp. 345–405.

[5] Ranieri Baraglia et al. «GoDel: Delaunay overlays in P2P networks via Gossip». In: *12th IEEE International Conference on Peer-to-Peer Computing, P2P 2012, Tarragona, Spain, September 3-5, 2012.* IEEE, 2012, pp. 1–12.

[6] Mark de Berg et al. *Computational Geometry: Algorithms and Applications.* 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[7] Prosenjit Bose et al. «On the Spanning Ratio of Gabriel Graphs and beta-Skeletons». In: *SIAM J. Discrete Math.* 20.2 (2006), pp. 412–427.

[8] Eliya Buyukkaya e Maha Abdallah. «Efficient triangulation for P2P networked virtual environments». In: *Multimedia Tools Appl.* 45.1-3 (2009), pp. 291–312.

[9] Siu-Wing Cheng, Tamal K. Dey e Jonathan Richard Shewchuk. *Delaunay Mesh Generation.* Chapman and Hall / CRC computer and information science series. CRC Press, 2013.

[10] Tanzeem Choudhury et al., cur. *Location and Context Awareness, 4th International Symposium, LoCA 2009, Tokyo, Japan, May 7-8, 2009, Proceedings*. Vol. 5561. Lecture Notes in Computer Science. Springer, 2009. ISBN: 978-3-642-01720-9. DOI: `10.1007/978-3-642-01721-6`. URL: `http://dx.doi.org/10.1007/978-3-642-01721-6`.

[11] Paolo Cignoni, Claudio Montani e Roberto Scopigno. «DeWall: A fast divide and conquer Delaunay triangulation algorithm in Ed». In: *Computer-Aided Design* 30.5 (1998), pp. 333–341.

[12] Jie Gao et al. «Geometric spanners for routing in mobile networks». In: *IEEE Journal on Selected Areas in Communications* 23.1 (2005), pp. 174–185.

[13] Luis Garcés-Erice et al. «Topology-Centric Look-Up Service». In: *Group Communications and Charges; Technology and Business Models, 5th COST264 International Workshop on Networked Group Communications, NGC 2003, and 3rd International Workshop on Internet Charging and QoS Technologies, ICQT 2003, Munich, Germany, September 16-19, 2003, Proceedings*. A cura di Burkhard Stiller et al. Vol. 2816. Lecture Notes in Computer Science. Springer, 2003, pp. 58–69.

[14] Mohsen Ghaffari, Behnoosh Hariri e Shervin Shirmohammadi. «A delaunay triangulation architecture supporting churn and user mobility in MMVEs». In: *Network and Operating System Support for Digital Audio and Video, 19th International Workshop, NOSSDAV 2009, Williamsburg, VA, USA. June 3-5, 2009, Proceedings*. A cura di Wei Tsang Ooi e Dongyan Xu. ACM, 2009, pp. 61–66.

[15] Leonidas J. Guibas e Jorge Stolfi. «Primitives for the Manipulation of General Subdivisions and Computation of Voronoi Diagrams». In: *ACM Trans. Graph.* 4.2 (1985), pp. 74–123.

[16] Nicholas J. A. Harvey et al. «SkipNet: A Scalable Overlay Network with Practical Locality Properties». In: *4th USENIX Symposium on Internet Technologies and Systems, USITS'03, Seattle, Washington, USA, March 26-28, 2003*. A cura di Steven D. Gribble. USENIX, 2003.

[17] Victoria J. Hodge e Jim Austin. «A binary neural k-nearest neighbour technique». In: *Knowl. Inf. Syst.* 8.3 (2005), pp. 276–291.

[18] Shun-Yun Hu, Jui-Fa Chen e Tsu-Han Chen. «VON: a scalable peer-to-peer network for virtual environments». In: *IEEE Network* 20.4 (2006), pp. 22–31.

[19] M. Frans Kaashoek e David R. Karger. «Koorde: A Simple Degree-Optimal Distributed Hash Table». In: *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*. A cura di M. Frans Kaashoek e Ion Stoica. Vol. 2735. Lecture Notes in Computer Science. Springer, 2003, pp. 98–107.

[20] Evangelos Kranakis, Harvinder Singh e Jorge Urrutia. «Compass routing on geometric networks». In: *Proceedings of the 11th Canadian Conference on Computational Geometry, UBC, Vancouver, British Columbia, Canada, August 15-18, 1999.* 1999.

[21] Dong-Young Lee e Simon S. Lam. «Efficient and accurate protocols for distributed delaunay triangulation under churn». In: *Proceedings of the 16th annual IEEE International Conference on Network Protocols, 2008. ICNP 2008, Orlando, Florida, USA, 19-22 October 2008.* IEEE Computer Society, 2008, pp. 124–136.

[22] Dong-Young Lee e Simon S. Lam. «Protocol Design for Dynamic Delaunay Triangulation». In: *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada.* IEEE Computer Society, 2007, p. 26.

[23] Xiang-Yang Li, Gruia Călinescu e Peng-Jun Wan. «Distributed Construction of Planar Spanner and Routing for Ad Hoc Wireless Networks». In: *Proceedings IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA, June 23-27, 2002.* IEEE, 2002.

[24] Jörg Liebeherr, Michael Nahas e Weisheng Si. «Application-layer multicasting with Delaunay triangulation overlays». In: *IEEE Journal on Selected Areas in Communications* 20.8 (2002), pp. 1472–1488.

[25] Marzolla M. *Simulating overlay networks with PeerSim.* URL: `http://www.cs.unibo.it/~babaoglu/courses/cas09-10/slides/peersim.pdf`.

[26] Giri Narasimhan e Michiel H. M. Smid. *Geometric spanner networks.* Cambridge University Press, 2007.

[27] Masaaki Ohnishi, Masugi Inoue e Hiroaki Harai. «Incremental Distributed Construction Method of Delaunay Overlay Network on Detour Overlay Paths». In: *JIP* 21.2 (2013), pp. 216–224.

[28] Masaaki Ohnishi, Ryo Nishide e Shinichi Ueshima. «Incremental Construction of Delaunay Overlaid Network for Virtual Collaborative Space». In: *3rd Conference on Creating, Connecting and Collaborating through Computing (C^5 2005), January 28-29, 2005, Cambridge, MA, USA.* IEEE Computer Society, 2005, pp. 75–82.

[29] *PeerSim Simulator.* URL: `http://peersim.sourceforge.net`.

[30] Sylvia Ratnasamy et al. «A scalable content-addressable network». In: *SIGCOMM.* 2001, pp. 161–172.

[31] S. Rebay. «Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm». In: *J. of Computational Physics* 106 (1993), pp. 125–138.

[32] Antony I. T. Rowstron e Peter Druschel. «Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems». In: *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings.* A cura di Rachid Guerraoui. Vol. 2218. Lecture Notes in Computer Science. Springer, 2001, pp. 329–350.

[33] Gwendal Simon, Moritz Steiner e Ernst Biersack. *Distributed Dynamic Delaunay Triangulation in d-Dimensional Spaces.* Rapp. tecn. 2005.

[34] Ion Stoica et al. «Chord: A scalable peer-to-peer lookup service for internet applications». In: *SIGCOMM.* 2001, pp. 149–160.

[35] Ivan Stojmenovic e Xu Lin. «Power-Aware Localized Routing in Wireless Networks». In: *IEEE Trans. Parallel Distrib. Syst.* 12.11 (2001), pp. 1122–1133.

[36] Godfried T. Toussaint. «The Relative Neighbourhood Graph of a Finite Planar Set». In: *Pattern Recognition* 12 (1980), pp. 261–268.

[37] *Triangulation Algorithms and Data Structures.* URL: `http://www.cs.cmu.edu/~quake/tripaper/triangle2.html`.

[38] Shinji Tsuboi et al. «Generating Skip Delaunay Network for P2P Geocasting». In: *Sixth International Conference on Creating, Connecting and Collaborating through Computing (C^5 2008), January 14-16, Poitiers, France.* IEEE Computer Society, 2008, pp. 179–186.

[39] Spyros Voulgaris, Daniela Gavidia e Maarten van Steen. «CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays». In: *J. Network Syst. Manage.* 13.2 (2005), pp. 197–217.

[40] Chinh T. Vu e Yingshu Li. «Delaunay-Triangulation Based Complete Coverage in Wireless Sensor Networks». In: *Seventh Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2009), 9-13 March 2009, Galveston, TX, USA.* IEEE Computer Society, 2009, pp. 1–5.

[41] Jiong Wang e Sirisha Medidi. «Energy Efficient Coverage with Variable Sensing Radii in Wireless Sensor Networks». In: *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2007, White Plains, New York, USA, 8-10 October 2007, Proceedings.* IEEE Computer Society, 2007, p. 61.

[42] Duncan J. Watts e Steven H. Strogatz. «Collective dynamics of /'small-world/' networks». In: *Nature* 393.6684 (giu. 1998), pp. 440–442. URL: `http://dx.doi.org/10.1038/30918`.

[43] Chun-Hsien Wu, Kuo-Chuan Lee e Yeh-Ching Chung. «A Delaunay Triangulation based method for wireless sensor network deployment». In: *Computer Communications* 30.14-15 (2007), pp. 2744–2752.

[44] Zhichen Xu e Zheng Zhang. *Building Low-maintenance Expressways for P2P Systems.* Rapp. tecn. 2002.

[45] Ben Y. Zhao, John Kubiatowicz e Anthony D. Joseph. «Tapestry: a fault-tolerant wide-area application infrastructure». In: *Computer Communication Review* 32.1 (2002), p. 81.